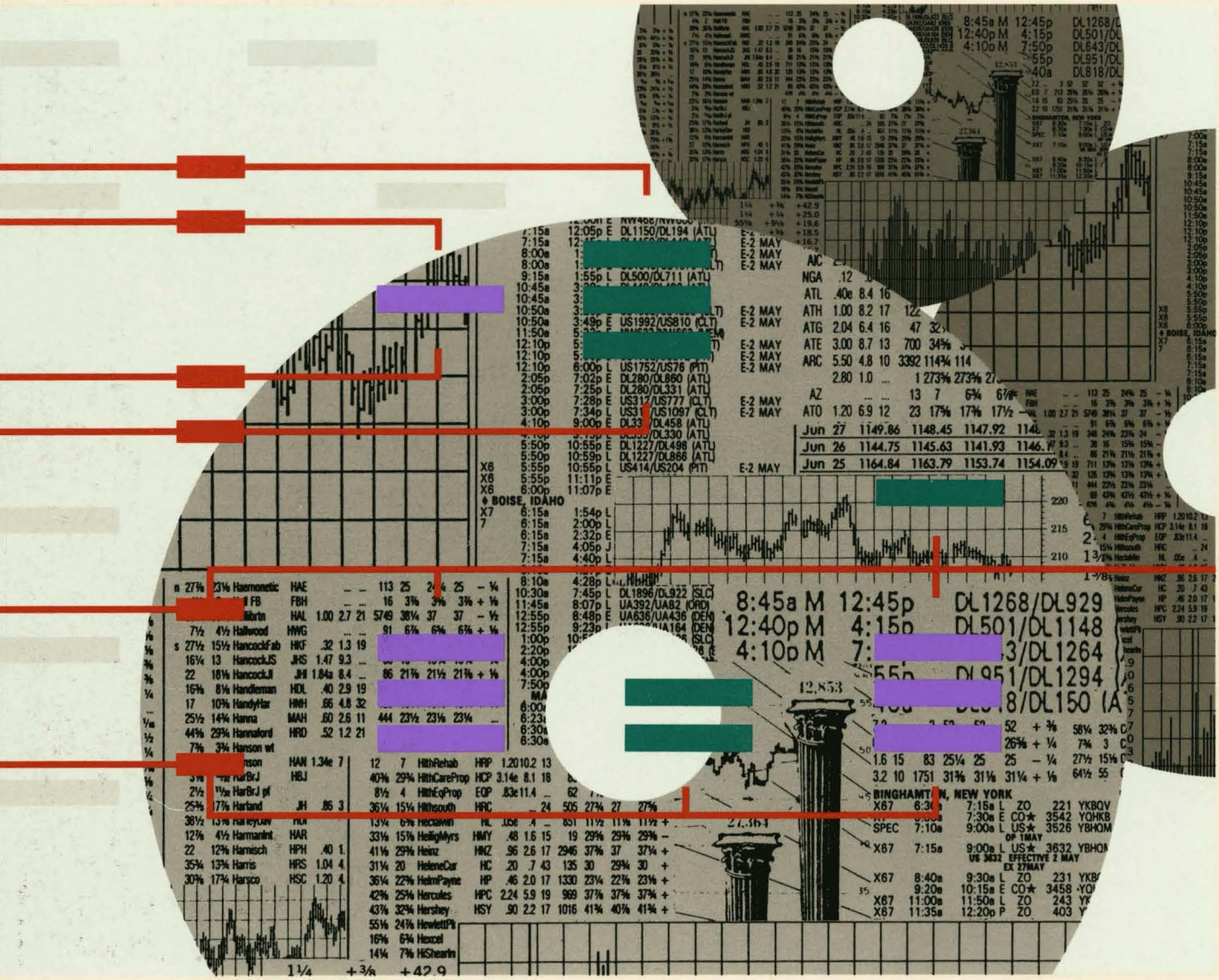
- *Availability in VAXcluster Systems*
- *Network Performance and Adapters*

Digital Technical Journal

Digital Equipment Corporation



Volume 3 Number 3

Summer 1991

The *Digital Technical Journal* is published quarterly by Digital Equipment Corporation, 146 Main Street MLO1-3/B68, Maynard, Massachusetts 01754-2571. Subscriptions to the *Journal* are $40.00 for four issues and must be prepaid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to the *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically to DTJ@CRL.DEC.COM. Single copies and back issues are available for $16.00 each from Digital Press of Digital Equipment Corporation, 12 Crosby Drive, Bedford, MA 01730-1493.

Digital employees may send subscription orders on the ENET to RDVAX::JOURNAL or by interoffice mail to mailstop MLO1-3/B68. Orders should include badge number, site location code, and address. All employees must advise of changes of address.

Comments on the content of any paper are welcomed and may be sent to the editor at the published-by or network address.

ISSN 0898-901X

Documentation Number EY-H890E-DP

Book production was done by Digital's Database Publishing Group in Northboro, MA.

Cover Design

Our cover graphic represents the shadowing, or replication, of data on multiple physical disks in a VAXcluster environment. VMS host-based volume shadowing provides the high data availability required for applications such as transaction processing and is the subject of one of the papers in this issue.

The cover was designed by Sandra Calef of Calef Associates.

Contents

5 ***Foreword***
Howard H. Hayakawa and George S. Hoff

Availability in VAXcluster Systems
Network Performance and Adapters

7 ***Design of VMS Volume Shadowing Phase II—Host-based Shadowing***
Scott H. Davis

16 ***Application Design in a VAXcluster System***
William E. Snaman, Jr.

27 ***New Availability Features of Local Area VAXcluster Systems***
Lee Leahy

36 ***Design of the DEC LANcontroller 400 Adapter***
Richard E. Stockdale and Judy B. Weiss

48 ***The Architecture and Implementation of a High-performance FDDI Adapter***
Satish L. Rege

64 ***Performance Analysis of a High-speed FDDI Adapter***
Ramsesh S. Kalkunte

78 ***Performance Analysis of FDDI***
Raj Jain

Editor's Introduction

Jane C. Blake
Editor

This issue of the *Digital Technical Journal* contains a collection of papers on two general topics—VAXcluster systems, and network adapters and performance. The first set of three papers describes new VMS VAXcluster developments and features; the second set addresses the topics of LAN adapter design and performance measurement techniques. A common theme across these papers is the development of technologies for interconnecting systems that offer high data availability without sacrificing performance.

VMS Volume Shadowing, described by Scott Davis, is a means of ensuring data availability and integrity in VMS VAXcluster systems. By maintaining multiple copies of data on separate devices, the volume shadowing technique protects data from being lost as the result of media deterioration or device failures. Scott discusses the advantages of the new design over controller-based shadowing and explains how this fully distributed software makes a broad range of topologies suitable for shadowing.

The growth capabilities and availability of VMS VAXcluster systems are characteristics well suited to applications with high-availability requirements. Sandy Snaman first presents an overview of the VAXcluster system architecture, including explanations of the layers, their purpose and function. He then gives practical insights into how the system implementation affects application design and reviews the choices available to application designers in the areas of client-server computing and data sharing.

The availability of applications and cluster configurations is also enhanced by developments in a new release of the VMS operating system. Lee Leahy describes a VMS feature that enables fail-over between multiple LAN adapters and compares this approach to a single-adapter implementation. He then discusses and gives examples of VMS features for network delay detection and reduction, and failure analysis in local area VAXcluster systems.

The focus then moves from VMS-level concerns to the design of network adapters and performance measurement. The adapter described by Dick Stockdale and Judy Weiss is the DEC LANcontroller 400, which connects systems based on Digital's XMI bus to an Ethernet LAN. This particular design improves on previous designs by transforming the adapter from a dumb to an intelligent adapter which can off-load the host. Consequently, the adapter supports systems that utilize the full bandwidth of Ethernet. The authors provide a system overview, performance metrics, and a critical examination of firmware-based design.

Like the LANcontroller 400, the FDDIcontroller 400 is an adapter that interfaces XMI-based systems to a LAN. However, as Satish Rege relates, this adapter was required to transmit data 30 times faster than Ethernet adapters. Satish discusses the architecture and the choices designers made to address the problem of interfacing a parallel high-bandwidth CPU bus (XMI) to a serial fiber-optic network bus (FDDI). Their design choices included a three-stage pipeline approach to buffering that enables these stages to proceed in an asynchronous fashion.

To ensure that the performance goals for the FDDIcontroller would be met, a simulation model was created. In his paper, Ram Kalkunte details the modeling methodology, reviews components, and presents simulation results. Ram describes how in addition to performance projections, the model provided designers with buffer sufficiency analysis and helped engineers analyze the functional correctness of the adapter design.

The high level of performance achieved by the FDDIcontroller was driven by the high performance of the FDDI LAN itself—100 megabits per second. Raj Jain's subject is performance measurement at the level of the FDDI LAN. Raj describes the performance analysis of Digital's implementation of FDDI and how various parameters affect system performance. As part of his presentation of the modeling and simulation methods used, he shares guidelines for setting the value of one of the key parameters, target token rotation time, to optimize performance. Raj has recently published a book on computer systems performance analysis, which is reviewed in the Further Readings section of this issue.

Jane Blake

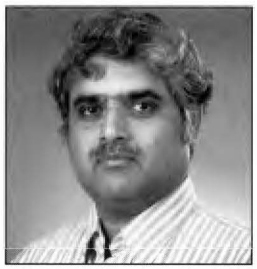
Biographies

Scott H. Davis Consultant software engineer Scott Davis is the VMS Cluster Technical Director. He is involved in future VMS I/O subsystem, VAXcluster, and storage strategies and led the VMS Volume Shadowing Phase II and VMS mixed-interconnect VAXcluster I/O projects. Since joining Digital in 1979, Scott worked on RT-11 development and led various real-time operating systems projects. He holds a B.S. (1978, cum laude) in computer science and applied mathematics from the State University of New York at Albany. Scott is a coinventor for four patents on the shadowing design.

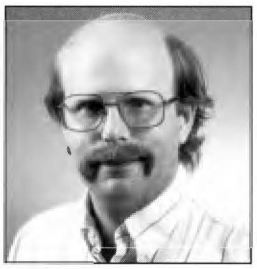
Raj Jain Raj Jain is a senior consulting engineer involved in the performance modeling and analysis of computer systems and networks, including VAXcluster, DECnet, Ethernet, and FDDI products. He holds a Ph.D. (1978) from Harvard University and has taught courses in performance analysis at M.I.T. A member of the Authors' Guild and senior member of IEEE, Raj has written over 25 papers and is listed in *Who's Who in the Computer Industry,* 1989. He is the author of *The Art of Computer Systems Performance Analysis* published recently by Wiley.

Ram Kalkunte As a member of the VAX VMS Systems and Servers Group, senior engineer Ram Kalkunte worked on the development of the high-performance XMI-to-FDDI adapter. Since joining Digital in 1987, he has also worked on various performance analysis and modeling projects. Ram received a B.E. (1984) in instrumentation engineering from Mysore University in India and an M.S. (1987) in electrical engineering from Worcester Polytechnic Institute. He has a pending patent related to dynamically arbitrating conflicts in a multiport memory controller.

Lee Leahy Lee Leahy is a principal software engineer in the VAXcluster group in ALPHA/EVMS Development. He is currently the project leader of the Local Area VAXcluster development effort and was responsible for the final design and implementation of the multiple-adapter version of PEDRIVER. Lee joined Digital in 1988 from ECAD. He is coauthor of the book *VMS Advanced Device Driver Techniques* and has been writing VMS device drivers since 1980. Lee received a B.S. degree in electrical engineering from Lehigh University in 1977.



Satish L. Rege Satish Rege joined Digital in 1977 as a member of the Mass Storage Group. He wrote the first proposal for the MSCP protocol and evaluated disk controller design alternatives (e.g., seek algorithms and disk caching) for the HSC50 by implementing architectural and performance simulation. He was instrumental in architecting the low-end controllers used in RF-series disks. His latest project was the high-performance FDDI adapter. Satish is a consulting engineer and received his Ph.D. from Carnegie-Mellon University, Pittsburgh.



William E. Snaman Principal software engineer Sandy Snaman joined Digital in 1980. He is the technical supervisor for the VAXcluster executive services area and a project leader for VAXcluster advanced development in the VMS Development Group. His group is responsible for ongoing development of the VMS lock manager, connection manager, and clusterwide services. Sandy teaches computer science at Daniel Webster College and developed and taught VAXcluster courses in Educational Services. He holds a B.S. (1985, magna cum laude) and an M.S.C.S. from the University of Lowell.

Richard E. Stockdale As a member of the Midrange Systems Engineering Group, Dick Stockdale was firmware project leader for the DEMNA project and prior to that for the DEBNA and DEBNI Ethernet adapters. He is currently a software consulting engineer working on LAN drivers in the VMS Development Group. Dick joined Digital in 1978 and performed diagnostic testing for 36-bit systems. He graduated from Worcester Polytechnic Institute in 1973 with a B.S. (magna cum laude) in computer science and a minor in electrical engineering. He is a member of Tau Beta Pi.

Judy B. Weiss Judy Weiss contributed to the design, implementation, debug, and performance analysis of the DEMNA Ethernet adapter as a member of the firmware team. She is a senior engineer working in the Data Center Systems and Servers Group as a gate array designer. Concurrently with the DEMNA, she worked on the firmware for the DEBNI adapter. Judy joined Digital in 1986 after receiving her B.S. (magna cum laude) in computer engineering from Boston University. She is a member of Tau Beta Pi and the Society for Women Engineers.

Foreword

Howard H. Hayakawa
Manager, VMS I/O and Cluster Development

George S. Hoff
Group Manager, High-availability Systems

Beginning as a vision for a highly available and expandable computing environment, Digital's VAXcluster system is today recognized across the industry as the premiere foundation for creating high-availability applications. The large number of VAXcluster sites and the range of their use testifies to the wide appeal of the capabilities of VAXcluster systems. Over 11,000 VAXcluster sites based on Digital's Computer Interconnect (CI) are being used in such diverse applications as manufacturing operations, banking, and telephone information systems. Sites based on the Digital Storage System Interconnect (DSSI) and Ethernet are even more numerous. A scan of software licenses shows an amazing acceptance of VAXcluster technology—more than 200,000 VAXcluster licenses have been sold to date.

Built from standard processors and a general-purpose operating system, a VAXcluster system is a loosely coupled, highly integrated configuration of VAX VMS processors and storage systems that operates as a single system. Significantly, VAXcluster systems are so well integrated that users are often not aware they are using a distributed system. In addition to the benefits of tight integration, these configurations provide Digital's customers with the flexibility to easily expand and with the features needed for high-availability applications.

Started in 1984, VAXcluster systems were limited to specialized, proprietary interconnects and storage servers, which restricted them to the confines of a single computer room. In 1989, the cluster system was extended to support both industry-standard SCSI (small computer systems interface) storage and Digital's DSSI storage interconnect. Today, VAXcluster systems support a wide range of communication interconnects, including CI and DSSI, and industry-standard local area networks such as Ethernet and FDDI. Storage systems now supported cover the spectrum from standard, economical SCSI peripherals to high-performance RA-series drives for large configurations. This well-architected system has allowed for expansion across an ever wider geography: from room to building to multiple buildings. Moreover, the entire range of VAX processors—from VAXstation workstations to VAX 9000 mainframes—are supported. The tight integration, flexibility, and power of today's VAXcluster systems is unparalleled.

The VAXcluster architecture which Digital initiated in the 1980s continues to encompass new advances and innovative technologies that ensure data availability and integrity. This issue of the *Digital Technical Journal* presents several new VMS VAXcluster products and features, and complementary developments in the areas of network adapters and performance. One of the products described is VMS Volume Shadowing Phase II which permits users to place redundant data on separate storage devices where most appropriate within the system, thus dramatically increasing the availability potential of VAXcluster systems. A paper on multi-rail local area VAXclusters shows how customers are now able to add parallel LAN connections to increase network capacity and to survive failure of a network connection. With shadowing and multiple communication paths, recovery from site failure need no longer incur the delays associated with restoration from archives.

Just as the VAXcluster software was able to exploit the Ethernet to extend capabilities throughout a building, it is now able to exploit the high performance and extent of an FDDI LAN.

The new industry-standard FDDI LAN allows the VAXcluster software to extend the system's range by a factor of 1,000. Papers on both an Ethernet adapter and an FDDI adapter describe the care taken to ensure that adapter performance matches that of the target processor, which is one of the keys to achieving maximum performance in the overall VAXcluster system. Performance of the FDDI LAN itself is also one of the topics included here. FDDI's performance and range permit for the first time the ability to create integrated, high-availability solutions that span multiple buildings. With combined FDDI and VMS VAXcluster technology, a bank's VAXcluster system can extend from a computer center in Manhattan to a standby center in New Jersey. Should Manhattan lose power, a disaster team can bring the bank's application into operation in New Jersey after only minutes. The days of waiting for archives or driving tapes and disks across the river are over.

Digital's VAX VMS, clusters, FDDI, and networking products continue to evolve; the process of integrating new technologies is ongoing. The papers in this issue describe the latest steps we have taken to extend the range and availability of VAXcluster systems. Future issues of the *Journal* will keep you apprised of the latest stages in this evolutionary process.

Scott H. Davis

Design of VMS Volume Shadowing Phase II— Host-based Shadowing

VMS Volume Shadowing Phase II is a fully distributed, clusterwide data availability product designed to replace the obsolete controller-based shadowing implementation. Phase II is intended to service current and future generations of storage architectures. In these architectures, there is no intelligent, multiunit controller that functions as a centralized gateway to the multiple drives in the shadow set. The new software makes many additional topologies suitable for shadowing, including DSSI drives, DSA drives, and shadowing across VMS MSCP servers. This last configuration allows shadow set members to be separated by any supported cluster interconnect, including FDDI. All essential shadowing functions are performed within the VMS operating system. New MSCP controllers and drives can optionally implement a set of shadowing performance assists, which Digital intends to support in a future release of the shadowing product.

Overview

Volume shadowing is a technique that provides data availability to computer systems by protecting against data loss from media deterioration, communication path failures, and controller or device failures. The process of volume shadowing entails maintaining multiple copies of the same data on two or more physical volumes. Up to three physical devices are bound together by the volume shadowing software and present a virtual device to the system. This device is referred to as a shadow set or a virtual unit. The volume shadowing software replicates data across the physical devices. All shadowing mechanisms are hidden from the users of the system, i.e., applications access the virtual unit as if it were a standard, physical disk. Figure 1 shows a VMS Volume Shadowing Phase II set for a Digital Storage Systems Interconnect (DSSI) configuration of two VAX host computers.

Product Goals

The VMS host-based shadowing project was undertaken because the original controller shadowing product is architecturally incompatible with many prospective storage devices and their connectivity requirements. Controller shadowing requires an intelligent, common controller to access all physical devices in a shadow set. Devices such as the RF-series integrated storage elements (ISEs) with DSSI adapters and the RZ-series small computer systems interface (SCSI) disks present configurations that conflict with this method of access.

To support the range of configurations required by our customers, the new product had to be capable of shadowing physical devices located anywhere within a VAXcluster system and of doing so in a controller-independent fashion. The VAXcluster I/O system provides parallel access to storage devices from all nodes in a cluster simultaneously. In order to meet its performance goals, our shadowing product had to preserve this semantic also. Figure 2 shows clusterwide shadow sets for a hierarchical storage controller (HSC) configuration with multiple computer interconnect (CI) buses. When compared to Figure 1, this figure shows a larger cluster containing several clusterwide shadow sets. Note that multiple nodes in the cluster have direct, writable access to the disks comprising the shadow sets.

In addition to providing highly available access to shadow sets from anywhere in a cluster, the new shadowing implementation had other requirements. Phase II had to deliver performance comparable to that of controller-based shadowing,

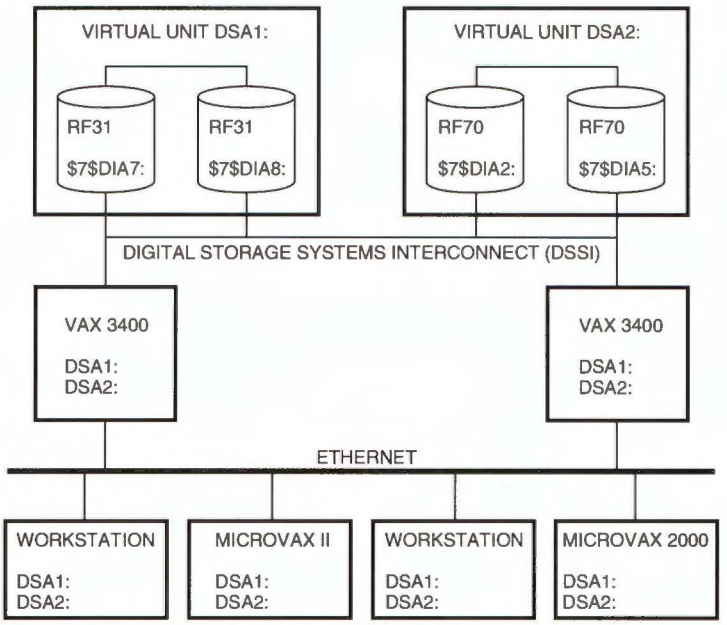


Figure 1 Phase II Shadow Set for a Dual-host DSSI Configuration

maximize application I/O availability, and ensure data integrity for critical applications.

In designing the new product, we benefited from customer feedback about the existing implementation. This feedback had a positive impact on the design of the host-based shadowing implementation. Our goals to maximize application I/O availability during transient states, to provide customizable, event-driven design and fail-over, to enable all cluster nodes to manage the shadow sets, and to enhance system disk capabilities were all affected by customer feedback.

Technical Challenges

To provide volume shadowing in a VAXcluster environment running under the VMS operating system required that we solve complex, distributed systems problems.[1] This section describes the most significant technical challenges we encountered and the solutions we arrived at during the design and development of the product.

Membership Consistency To ensure the level of integrity required for high availability systems, the shadowing design must guarantee that a shadow set has the same membership and states on all nodes in the cluster. A simple way to guarantee this property would have been a strict client-server implementation, where one VAX computer serves the shadow set to the remainder of the cluster. This approach, however, would have violated several design goals; the intermediate hop required by data transfers would decrease system performance, and any failure of the serving CPU would require a lengthy fail-over and rebuild operation, thus negatively impacting system availability.

To solve the problem of membership consistency, we used the VMS distributed lock manager through a new executive thread-level interface.[2,3] We designed a set of event-driven protocols that shadowing uses to guarantee membership consistency. These protocols allowed us to make the shadow set virtual unit a local device on all nodes in the cluster. Membership and state information about the shadow set is stored on all physical members in an on-disk data structure called the storage control block (SCB). One way that shadowing uses this SCB information is to automatically determine the most up-to-date shadow set member(s) when the set is created. In addition to distributed synchronization primitives, the VMS lock manager provides a capability for managing a distributed state variable called a lock value block. Shadowing uses the lock value block to define a disk that is guaranteed to be a current member of the shadow set. Whenever a membership change is made, all nodes take part in a protocol of lock operations; the value block and the on-disk SCB are the final arbiters of set constituency.

Sequential Commands A sequential I/O command, i.e., a Mass Storage Control Protocol (MSCP) concept, forces all commands in progress to complete before the sequential command begins execution. While a sequential command is pending, all new I/O requests are stalled until that sequential command completes execution. Shadowing requires the capability to execute a clusterwide, sequential command during certain operations. This capability, although a simple design goal for a client-server implementation, is a complex one for a distributed access model. We chose an event-driven, request/response protocol to create the sequential command capability.

Since sequential commands have a negative impact on performance, we limited the use of these commands to performing membership changes, mount/dismount operations, and bad block and merge difference repairs. Steady state processing never requires using sequential commands.

Full Copy A full copy is the means by which a new member of the shadow set is made current with the rest of the set. The challenge is to make copy operations unintrusive; application I/Os must proceed with minimal impact so that the level of service provided by the system is both acceptable and predictable. VMS file I/O provides record-level sharing through the application transparent locking provided by the VAX RMS software, Digital's record management services. Shadowing operates at the physical device level to handle a variety of low-level errors. Because shadowing has no knowledge of the higher-layer record locking, a copy operation must guarantee that the application I/Os and the copy operation itself generate the correct results and do so with minimal impact on application I/O performance.

Merge Operations Merge operations are triggered when a CPU with write access to a shadow set fails. (Note that with controller shadowing, merge operations are copy operations that are triggered when an HSC fails.) Devices may still be valid members of the shadow set but may no longer be identical, due to outstanding writes in progress when the host

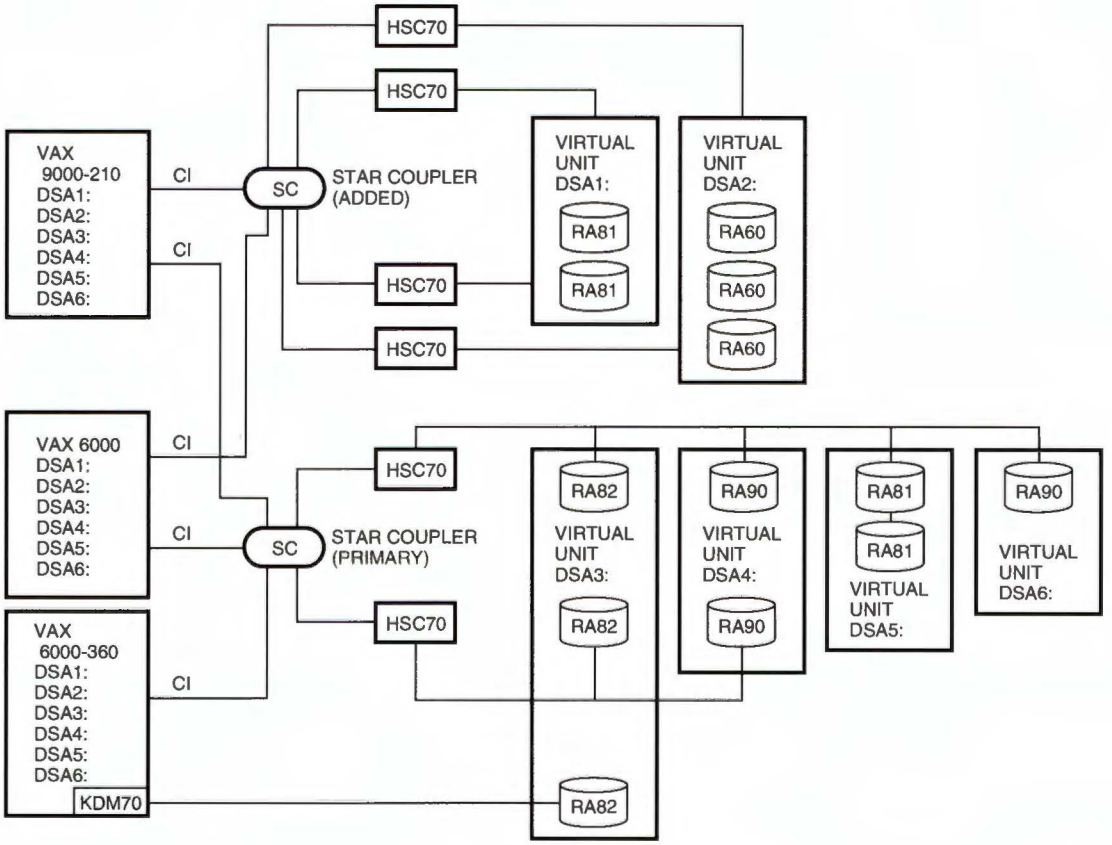


Figure 2 Clusterwide Shadow Sets for an HSC Configuration with Multiple CI Buses

CPU failed. The merge operation must detect and correct these differences, so that successive application reads for the same data produce consistent results. As for full copy operations, the challenge with merge processing is to generate consistent results with minimal impact on application I/O performance.

Booting and Crashing System disk shadowing presents some special problems because the shadow set must be accessible to CPUs in the cluster when locking protocols and inter-CPU communication are disabled. In addition, crashing must ensure appropriate behavior for writing crash dumps through the primitive bootstrap driver, including how to propagate the dump to the shadow set. It was not practical to modify the bootstrap drivers because they are stored in read-only memory (ROM) on various CPU platforms that shadowing would support.

Error Processing One major function of volume shadowing is to perform appropriate error processing for members of the shadow set, while maximizing data availability. To carry out this function, the software must prevent deadlocks between nodes and decide when to remove devices from the shadow set. We adopted a simple recovery ethic: a node that detects an error is responsible for fixing that error. Membership changes are serialized in the cluster, and a node only makes a membership change if the change is accompanied by improved access to the shadow set. A node never makes a change in membership without having access to some source members of the set.

Architecture

Phase II shadowing provides a local virtual unit on each node in the cluster with distributed control of that unit. Although the virtual unit is not served to the cluster, the underlying physical units that constitute a shadow set are served to the cluster using the standard VMS mechanisms. This scheme has many data availability advantages. The Phase II design

- Allows shadowing to use all the VMS controller fail-over mechanisms for physical devices. As a result, member fail-over approaches hardware speeds. Controller shadowing does not provide this capability.
- Allows each node in the cluster to perform error recovery based on access to physical data source members. The shadowing software treats communication failures between any cluster node and shadow set members as normal shadowing events with customer-definable recovery metrics.

Major Components

VMS Volume Shadowing Phase II consists of two major components: SHDRIVER and SHADOW_SERVER. SHDRIVER is the shadowing virtual unit driver. As a client of disk class drivers, SHDRIVER is responsible for handling all I/O operations that are directed to the virtual unit. SHDRIVER issues physical I/O operations to the disk class driver to satisfy the shadow set virtual unit I/O requests. SHDRIVER is also responsible for performing all distributed locking and for driving error recovery.

SHADOW_SERVER is a VMS ancillary control process (ACP) responsible for driving copy and merge operations performed on the local node. Only one optimal node is responsible for driving a copy or merge operation on a given shadow set, but when a failure occurs the operation will fail over and resume on another CPU. Several factors determine this optimal node including the types of access paths, and controllers for the members and user-settable, per-node copy quotas.

Primitives

This section describes the locking protocols and error recovery processing functions that are used by the shadowing software. These primitives provide basic synchronization and recovery mechanisms for shadow sets in a VAXcluster system.

Locking Protocols The shadowing software uses event-driven locking protocols to coordinate clusterwide activity. These request/response protocols provide maximum application I/O performance. A VMS executive interface to the distributed lock manager allows shadowing to make efficient use of locking directly from SHDRIVER.

One example of this use of locking protocols in VMS Volume Shadowing Phase II is the sequential command protocol. As mentioned in the Technical Challenges section, shadowing requires the sequential command capability but minimizes the use of this primitive. Phase II implements the capability by using several locks, as described in the following series of events.

A node that needs to execute a sequential command first stalls I/O locally and flushes operations

in progress. The node then performs lock operations that ensure serialization and sends sequential stall requests to other nodes that have the shadow set mounted. This initiating thread waits until all other nodes in the cluster have flushed their I/Os and responded to the node requesting the sequential operation. Once all nodes have responded or left the cluster, the operations that compose the sequential command execute. When this process is complete, the locks are released, allowing asynchronous threads on the other nodes to proceed and automatically resume I/O operations. The local node resumes I/O as well.

Error Recovery Processing Error recovery processing is triggered by either asynchronous notification of a communication failure or a failing I/O operation directed towards a physical member of the shadow set. Two major functions of error recovery are built into the virtual unit driver: active and passive volume processing.

Active volume processing is triggered directly by events that occur on a local node in the cluster. This type of volume processing uses a simple, localized ethic for error recovery from communication or controller failures. Shadow set membership decisions are made locally, based on accessibility. If no members of a shadow set are currently accessible from a node, then the membership does not change. If some but not all members of the set are accessible, the local node, after attempting failover, removes some members to allow application I/O to proceed. The system manager sets the time period during which members may attempt failover. The actual removal operation is a sequential command. The design allows for maximum flexibility and quick error recovery and implicitly avoids deadlock scenarios.

Passive volume processing responds to events that occur elsewhere in the cluster; messages from nodes other than the local one trigger the processing by means of the shadowing distributed locking protocols. This volume processing function is responsible for verifying the shadow set membership and state on the local node and for modifying this membership to reflect any changes made to the set by the cluster. To accomplish these operations, the shadowing software first reads the lock value block to find a disk guaranteed to still be in the shadow set. Then the recovery process retrieves the physical member's on-disk SCB data and uses this information to perform the relevant data structure updates on the local node.

Application I/O requests to the virtual unit are always stalled during volume processing. In the case of active volume processing, the stalling is necessary because many I/Os would fail until the error was corrected. In passive volume processing, the I/O requests are stalled because the membership of the set is in doubt, and correct processing of the request cannot be performed until the situation is corrected.

Steady State Processing

The shadowing virtual unit driver receives application read and write requests and must direct the I/O appropriately. This section describes these steady state operations.

Read Algorithms

The shadowing virtual unit driver receives application read requests and directs a physical I/O to an appropriate member of the set. SHDRIVER attempts to direct the I/O to the optimum device based on locally available data. This decision is based on (1) the access path, i.e., local or served by the VMS operating system, (2) the service queue lengths at the candidate controller, and (3) a round-robin algorithm among equal paths. Figure 3 shows a shadow set read operation. An application read to the shadow set causes a single physical read to be sent to an optimal member of the set. In Figure 3, there is one local and one remote member, so the read is sent to the local member.

Data repair operations caused by media defects are triggered by a read operation failing with an appropriate error, such as forced error or parity. The shadowing driver attempts this repair using

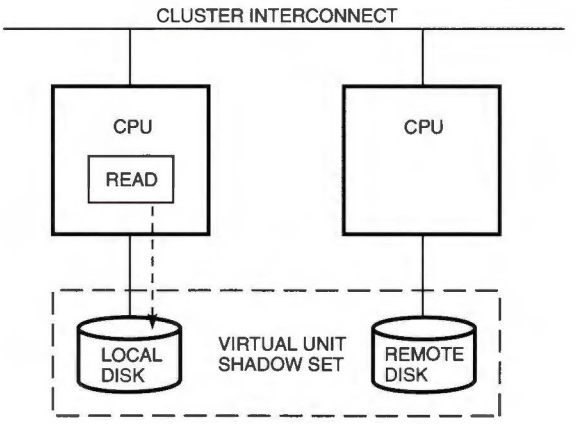


Figure 3 Shadow Set Read Operation

another member of the shadow set. This repair operation is performed with the synchronization of a sequential command. Sequential protection is required because a read operation is being converted into a write operation without explicit, RMS-layer synchronization.

Write Algorithms

The shadowing virtual unit driver receives application write requests and then issues, in parallel, write requests to the physical members of the set. The virtual unit write operation does not complete until all physical writes complete. A shadow set write operation is shown in Figure 4. Physical write operations to member units can fail or be timed out; either condition triggers the shadowing error recovery logic and can cause a fail-over or the removal of the erring device from the shadow set.

Transient State Processing

Shadowing performs a variety of operations in order to maintain consistency among the members of the set. These operations include full copy, merge, and data repair and recovery. This section describes these transient state operations.

Full Copy

Full copy operations are performed under direct system manager control. When a disk is added to the shadow set, copy operations take place to make the contents of this new set member identical to that of the other members. Copy operations are transparent to application processing. The new member of the shadow set does not provide any data availability protection until the copy completes.

There is no explicit gatekeeping during the copy operation. Thus, application read and write operations occur in parallel with copy thread reads and writes. As shown in Figure 5, correct results are accomplished by the following algorithm. During the full copy, the shadowing driver processes application write operations in two groups: first, those directed to all source members and second, writes to all full copy targets. The copy thread performs a sequence of read source, compare target, and write target operations on each logical block number (LBN) range until the compare operation succeeds. If an LBN range has such frequent activity that the compare fails many times, SHDRIVER performs a synchronized update. A distributed fence provides a clusterwide boundary between the copied and the uncopied areas of the new member. This fence is used to avoid performing the special full copy mechanisms on application writes to that area of the disk already processed by the copy thread.

This algorithm meets the goal of operational correctness (both the application and the copy thread achieve the proper results with regard to the contents of the shadow set members) and requires no synchronization with the copy thread. Thus, the algorithm achieves maximum application I/O availability during the transient state. Crucial to achieving this goal is the fact that, by design, the copy thread does not perform I/O optimization techniques such as double buffering. The copy operations receive equal service as application I/Os.

Merge Operations

The VMS Volume Shadowing Phase II merge algorithm meets the product goals of operational

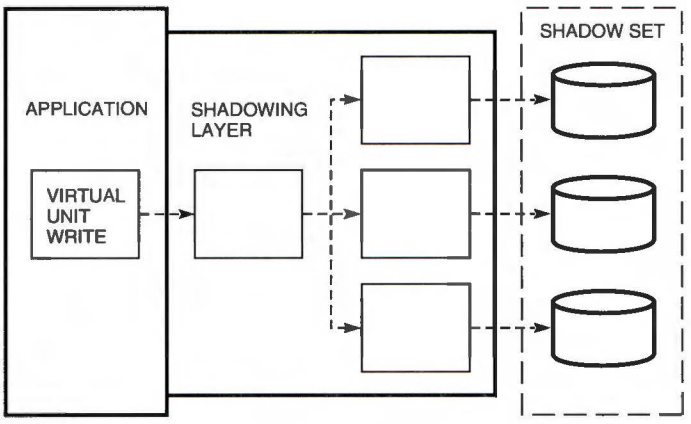
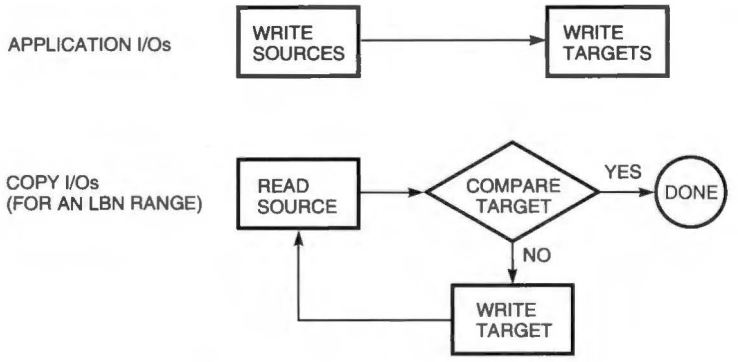


Figure 4 Shadow Set Write Operation



Note: No synchronization exists between the application and copy operations. I/Os can occur in parallel on different nodes in the cluster. Regardless of how the operations overlap, the correct data is copied to the target.

Figure 5 Full Copy Algorithm

correctness, while maintaining high application I/O availability and minimal synchronization. A merge operation is required when a CPU crashes with the shadow set mounted for write operations. A merge is needed to correct for the possibility of partially completed writes that may have been outstanding to the physical set members when the failure occurred. The merge operation ensures that all members contain identical data, and thus the shadow set virtual unit behaves like a single, highly available disk. It does not matter which data is more recent, only that the members are the same. This satisfies the purpose of shadowing, which is to provide data availability. But since the failure occurred while a write operation was in progress, this consistent shadow set can contain either old or new data. To make sure that the shadow set contains the most recent data, a data integrity technique such as journaling must be employed.

In Phase II shadowing, merge processing is distinctly different from copy processing. The shadow set provides full availability protection during the merge. As a result, merge processing is intentionally designed to be a background activity and to maximize application I/O throughput while the merge is progressing. The merge thread carefully monitors I/O rates and inserts a delay between its I/Os if it detects contention for shared system resources, such as adapters and interconnects.

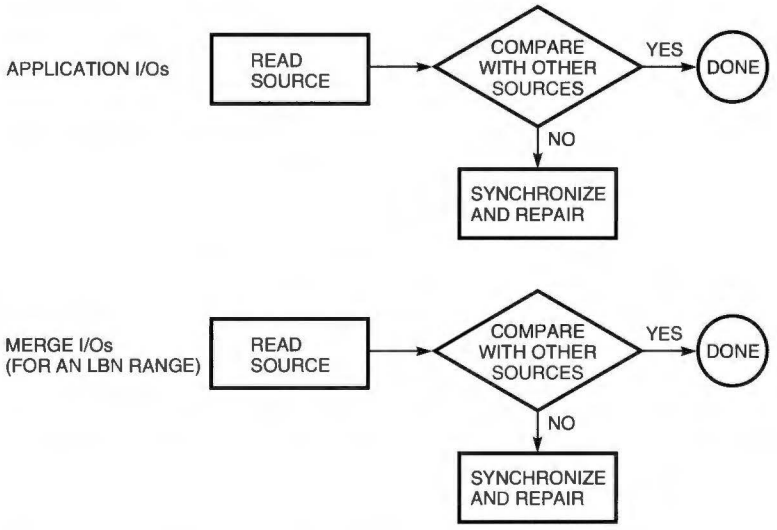
In addition to maximizing I/O availability, the merge algorithm is designed to minimize synchronization with application I/Os and to identify and correct data inconsistencies. Synchronization takes place only when a rare difference is found. When an application read operation is issued to a shadow set in the merge state, the set executes the read with merge semantics. Thus, a read to a source and a parallel compare with the other members of the set are performed. Usually the compare matches and the operation is complete. If a mismatch is detected, a sequential repair operation begins. The merge thread scans the entire disk in the same manner as the read, looking for differences. A distributed fence is used to avoid performing merge mechanisms for application reads to that area of the disk already processed by the merge thread. Figure 6 illustrates the merge algorithm.

Note that controller shadowing performs an operation called a merge copy. Although this HSC merge copy operation is designed for the same purpose as the Phase II operation, the approaches differ greatly. An HSC merge copy is triggered when an HSC, not a shadow set, fails and performs a copy operation; the HSC merge copy does not detect differences.

Performance Assists

A future version of the shadowing product is intended to utilize controller performance assists to improve copy and merge operations. These assists will be used automatically, if supported by the controllers involved in accessing the physical members of a shadow set.

COPY_DATA is the ability of a host to control a direct disk-to-disk transfer without the data entering or leaving the host CPU I/O adapters and memory. This capability will be used by full copy processing to decrease the system impact, the



Note: Infrequent synchronization exists between the application and merge operations. I/Os can occur in parallel on different nodes in the cluster. Regardless of how the operations overlap, data integrity is preserved.

Figure 6 Merge Algorithm

bandwidth, and the time required for a full copy. The members of the set and/or their controllers must share a common interconnect in order to use this capability. The COPY_DATA operation performs specific shadowing around the active, copy LBN range to ensure correctness. This operation involves LBN range-based gatekeeping in the copy target device controller.

Controller write logging is a future capability in controllers, such as HSCs, that will allow more efficient merge processing. Shadowing write operation messages will include information for the controller to log I/Os in its memory. These logs will then be used by the remaining host CPUs during merge processing to determine exactly which blocks contain outstanding write operations from a failed CPU. With such a performance assist, merge operations will take less time and will have less impact on the system.

Data Repair and Recovery

As discussed in the Primitives section, data repair operations are triggered by failing reads and are repaired as sequential commands. Digital Storage Architecture (DSA) devices support two primitive capabilities that are key to this repair mechanism. When a DSA controller detects a media error, the block in question is sometimes repaired automatically, thus requiring no shadowing intervention. When the controller cannot repair the data, a spare block is revectored to this LBN, and the contents of the block are marked with a forced error. This causes subsequent read operations to fail, since the contents of the block are lost.

The forced error returned on a read operation is the signal to the shadowing software to execute a repair operation. SHDRIVER attempts to read usable data from another source device. If such data is available, the software writes the data to the revectored block and then returns the data to the application. If no usable data source is available, the software performs write operations with a forced error to all set members and signals the application that this error condition has occurred. Note that a protected system buffer is used for this operation because the application reading the data may not have write access.

A future shadowing product is intended to support SCSI peripherals, which do not have the DSA primitives outlined above. There is no forced error indicator in the SCSI architecture, and the revector operation is nonatomic. To perform shadowing data repair on such devices, we will use the READL/WRITEL capability optionally supported by SCSI devices. These I/O functions allow blocks to be read and written with error correction code (ECC)

data. Shadowing emulates forced error by writing data with an intentionally incorrect ECC. To circumvent the lack of atomicity on the revector operation, a device being repaired is temporarily marked as a full copy target until the conclusion of the repair operation. If the CPU fails in the middle of a repair operation, the repair target is now a full copy target, which preserves correctness in the presence of these nonatomic operations.

System Disk

System disk shadow sets presented some unique design problems. The system disk must be accessed through a single bootstrap driver and hence, a single controller type. This access takes place when multihost synchronization is not possible. These two access modes occur during system bootstrap and during a crash dump write.

Shadowed Booting

The system disk must be accessed by the system initialization code executing on the booting node prior to any host-to-host communication. Since the boot drivers on many processors reside in ROM, it was impractical to make boot driver modifications to support system disk processing. To solve this problem, the system disk operations performed prior to the controller initialization routine of the system device driver are read-only. It is safe to read data from a clusterwide, shared device without synchronization when there is little or no risk of the data being modified by another node in the cluster. At controller initialization time, shadowing builds a read-only shadow set that contains only the boot member. Once locking is enabled, shadowing performs a variety of checks on the system disk shadow set to determine whether or not the boot is valid. If the boot is valid, shadowing turns the single-member, read-only set into a multimember, writable set with preserved copy states. If this node is joining an existing cluster, the system disk shadow set uses the same set as the rest of the cluster.

Crash Dumps

The primitive boot driver uses the system disk to write crash dumps when a system failure occurs. This driver only knows how to access a single physical disk in the shadow set. But since a failing node automatically triggers a merge operation on shadow sets mounted for write, we can use the merge thread to process the dump file. The merge occurs either when the node leaves the cluster (if there are other nodes present) or later, when the set is reformed. As the source for merge difference repairs, the merge process attempts to use the member to which the dump file was written and propagates the dump file to the remainder of the set. The mechanism here for dump file propagation is best-effort, not guaranteed; but since writing the dump is always best-effort, this solution is considered acceptable.

Conclusion

VMS Volume Shadowing Phase II is a state-of-the-art implementation of distributed data availability. The project team arrived at innovative solutions to problems attributable to a set of complex, conflicting goals. Digital has applied for four patents on various aspects of this technology.

Acknowledgments

I would like to acknowledge the efforts and contributions of the other members of the VMS shadowing engineering team: Renee Culver, William Goleman, and Wai Yim. In addition, I would also like to acknowledge Sandy Snaman for Fork Thread Locking, Ravindran Jagannathan for performance analysis, and David Thiel for general consulting.

References

1. N. Kronenberg, H. Levy, W. Strecker, and R. Merewood, "The VAXcluster Concept: An Overview of a Distributed System," *Digital Technical Journal* (September 1987): 7–21.
2. W. Snaman, Jr. and D. Thiel, "The VAX/VMS Distributed Lock Manager," *Digital Technical Journal* (September 1987): 29–44.
3. W. Snaman, Jr., "Application Design in a VAXcluster System," *Digital Technical Journal*, vol 3. no. 3 (Summer 1991, this issue): 16–26.

William E. Snaman, Jr.

Application Design in a VAXcluster System

VAXcluster systems provide a flexible way to configure a computing system that can survive the failure of any component. In addition, these systems can grow with an organization and can be serviced without disruption to applications. These features make VAXcluster systems an ideal base for developing high-availability applications such as transaction processing systems, servers for network client-server applications, and data sharing applications. Understanding the basic design of VAXcluster systems and the possible configuration options can help application designers take advantage of the availability and growth characteristics of these systems.

Many organizations depend on near constant access to data and computing resources; interruption of these services results in the interruption of primary business functions. In addition, growing organizations face the need to increase the amount of computing power available to them over an extended period of time. VAXcluster systems provide solutions to these data availability and growth problems that modern organizations face.[1]

This paper begins with an overview of VAXcluster systems and application design in such systems and proceeds with a detailed discussion of VAXcluster design and implementation. The paper then focuses on how this information affects the design of applications that take advantage of the availability and growth characteristics of a VAXcluster system.

Overview of VAXcluster Systems

VAXcluster systems are loosely coupled multiprocessor configurations that allow the system designer to configure redundant hardware that can survive most types of equipment failures. These systems provide a way to add new processors and storage resources as required by the organization. This feature eliminates the need either to buy nonessential equipment initially or to experience painful upgrades and application conversions as the systems are outgrown.

The VMS operating system, which runs on each processor node in a VAXcluster system, provides a high level of transparent data sharing and independent failure characteristics. The processors interact to form a cooperating distributed operating system. In this system, all disks and their stored files are accessible from any processor as if those files were connected to a single processor. Files can be shared transparently at the record level by application software.

To provide the features of a VAXcluster system, the VMS operating system was enhanced to facilitate this data sharing and the dynamic adjustment to changes in the underlying hardware configuration. These enhancements make it possible to dynamically add multiple processors, storage controllers, disks, and tapes to a VAXcluster system configuration. Thus, an organization can purchase a small system initially and expand as needed. The addition of computing and storage resources to the existing configuration requires no software modifications or application conversions and can be accomplished without shutting down the system. The ability to use redundant devices virtually eliminates single points of failure.

Application Design in a VAXcluster Environment

Application design in a VAXcluster environment involves making some basic choices. These choices concern the type of application to be designed and the method used to synchronize the events that occur during the execution of the application. The designer must also consider application communication within a VAXcluster system. A discussion of these issues follows.

General Choices for Application Design

This section briefly describes the general choices available to application designers in the areas of client-server computing and data access.

Client-server Computing The VAXcluster environment provides a fine base for client-server computing. Application designers can construct server applications that run on each node and accept requests from clients running on nodes in the VAXcluster system or elsewhere in a wider network.

If the node running a server application fails, the clients of that server can switch to another server running on a surviving node. The new server can access the same data on disk or tape that was being accessed by the server that failed. In addition, the redundancy offered by the VMS Volume Shadowing Phase II software eliminates data unavailability in the event of a disk controller or media failure.[2] The system is thus very available from the perspective of the client applications.

Access to Storage Devices Many application design questions involve how to best access the data stored on disk. One major advantage of the VAXcluster system design is that disk storage devices can be accessed from all nodes in an identical manner. The application designer can choose whether the access is simultaneous from multiple nodes or from one node at a time. Consequently, applications can be designed using either partitioned data access or shared data access.

Using a partitioned data model, the application designer can construct an application that limits data access to a single node or subset of the nodes. The application runs as a server on a single node and accepts requests from other nodes in the cluster and in the network. And because the application runs on a single node, there is no need to synchronize data access with other nodes. Eliminating this source of communication latencies can improve performance in many applications. Also, if synchronization is not required, the designer can make the best use of local buffer caches and can aggregate larger amounts of data for write operations, thus minimizing I/O activity.

An application that uses partitioned data access lends itself to many types of high-performance database and transaction processing environments. VAXcluster systems provide such an application with the advantage of having a storage medium that is available to all nodes even when they are not actively accessing the data files. Thus, if the server node fails, another server running on a surviving node can assume the work and be able to access the same files. For this type of application design, VAXcluster systems offer the performance advantages of a partitioned data model without the problems associated with the failure of a single server.

Using a shared data model, the application designer can create an application that runs simultaneously on multiple VAXcluster nodes, which naturally share data in a file. This type of application can prevent the bottlenecks associated with a single server and take advantage of opportunities for parallelism on multiple processors. The VAX RMS software can transparently share files between multiple nodes in a VAXcluster system. Also, Digital's database products, such as Rdb/VMS and VAX DBMS software, provide the same data-sharing capabilities. Servers running on multiple nodes of a VAXcluster system can accept requests from clients in the network and access the same files or databases. Because there are multiple servers, the application continues to function in the event that a single server node fails.

Application Synchronization Methods

The application designer must also consider how to synchronize events that take place on multiple nodes of a VAXcluster system. Two main methods can be used to accomplish this: the VMS lock manager and the DECdtm services that provide VMS transaction processing support.

VMS Lock Manager The VMS lock manager provides services that are flexible enough to be used by cooperating processes for mutual exclusion, synchronization, and event notification.[3] An application uses these services either directly or indirectly through components of the system such as the VAX RMS software.

DECdtm Services The VMS operating system provides a set of services to facilitate transaction processing.[4] These DECdtm services enable the application designer to implement atomic transactions either directly or indirectly. The services use a two-phase commit protocol. A transaction may span multiple nodes of a cluster or network. The support provided allows multiple resource managers, such as the VAX DBMS, Rdb/VMS, and VAX RMS software products, to be combined in a single transaction. The DECdtm transaction processing services take advantage of the guarantees against partitioning, the distributed lock manager, and the data availability features, all provided by VAXcluster systems.

VAXcluster and Networkwide Communication Services

Application communication between different processors in a VAXcluster system is generally accomplished using DECnet task-to-task communication services or other networking software such as the transmission control protocol (TCP) and the internet protocol (IP). Client-server applications or peer-to-peer applications are easy to develop with these services. The services allow processes to locate or start remote servers and then to exchange messages.

Since the individual nodes of a VAXcluster system exist as separate entities in a wider communication network, applications communication inside a VAXcluster system can rely on general network interfaces. Thus, no special-purpose communication services were developed. Applications are simpler to design when they can communicate within the cluster in the same manner in which they communicate with nodes located outside the VAXcluster system.

A DECnet feature known as cluster alias provides a collective name for the nodes in a VAXcluster system. Application software can connect to a node in the cluster using the cluster alias name rather than a specific node name. This feature frees the application from keeping track of individual nodes in the VAXcluster system and results in design simplification and configuration flexibility.

VAXcluster Design and Implementation Details

To understand how the design and implementation of a VAXcluster system affects application design, one must be familiar with the basic architecture of such a system, as shown in Figure 1. This section describes the layers, which range from the communication mechanisms to the users of the system.

Port Layer

The port layer consists of the lowest levels of the architecture, including a choice of communication ports and physical paths (buses). The VAXcluster software requires at least one logical communication pathway between each pair of processor nodes in the VAXcluster system. Several of the ports utilize multiple physical communication paths, which appear as a single logical path to the VAXcluster software. This redundancy provides better communication throughput and higher availability. If multiple logical paths exist between a pair of nodes, the VAXcluster software generally selects one for active use and relies on the remaining paths for backup in the event of failure.

The port layer can contain any of the following interconnects:

- Computer Interconnect (CI) bus
- Ethernet
- Fiber distributed data interface (FDDI)
- Digital Storage Systems Interconnect (DSSI) bus

Each bus is accessed by a port (also called an adapter) that connects to the processor node. For example the CI bus is accessed by way of a CI port. The various buses provide a wide spectrum of choices in terms of wire and adapter capacity, number of nodes that can be attached, distance between nodes, and cost.[5]

The CI bus was designed for access to storage and for reliable host-to-host communications. Each CI port connects to two redundant, high-speed physical paths. The CI port dynamically selects one of the two paths for each transmitted message. Messages are received on either path. Thus, two nodes can communicate on one path at the same time that two other nodes communicate on the other. If one physical path fails, the port simply uses the remaining path. The existence of the two physical paths is hidden from the software that uses the CI port services. From the standpoint of the cluster software, each port represents a single logical path to a remote node. Multiple CI ports can be used to provide multiple logical paths between pairs of nodes. An automatic load-sharing feature distributes the load between pairs of ports.

The DSSI bus was primarily designed for access to disk and tape storage. However, the bus has proven an excellent way to connect small numbers of processors using the VAXcluster protocols. Each DSSI port connects to a single high-speed physical path. As in the case of the CI bus, several DSSI ports may be connected to a node to provide redundant paths. (Note that the KFQSA DSSI port is for storage access only and provides no general communication service between nodes.)

Ethernet and FDDI are open local area networks, generally shared by a wide variety of consumers. Consequently, the VAXcluster software was designed to use the Ethernet and FDDI ports and buses simultaneously with the DECnet or TCP/IP protocols. This is accomplished by allowing the Ethernet data link software to control the hardware port. This

software provides a multiplexing function such that the cluster protocols are simply another user of a shared hardware resource.

Each Ethernet and FDDI port connects to a single physical path. There may be more than one port on each processor node. This means that there may be many separate paths between any pair of nodes when multiple ports are used. The port driver software combines the multiple Ethernet and FDDI paths into a single logical path between any pair of nodes. The load is automatically distributed among the various possible physical paths by an algorithm that chooses the best path in terms of adapter capacity and path latency.[6]

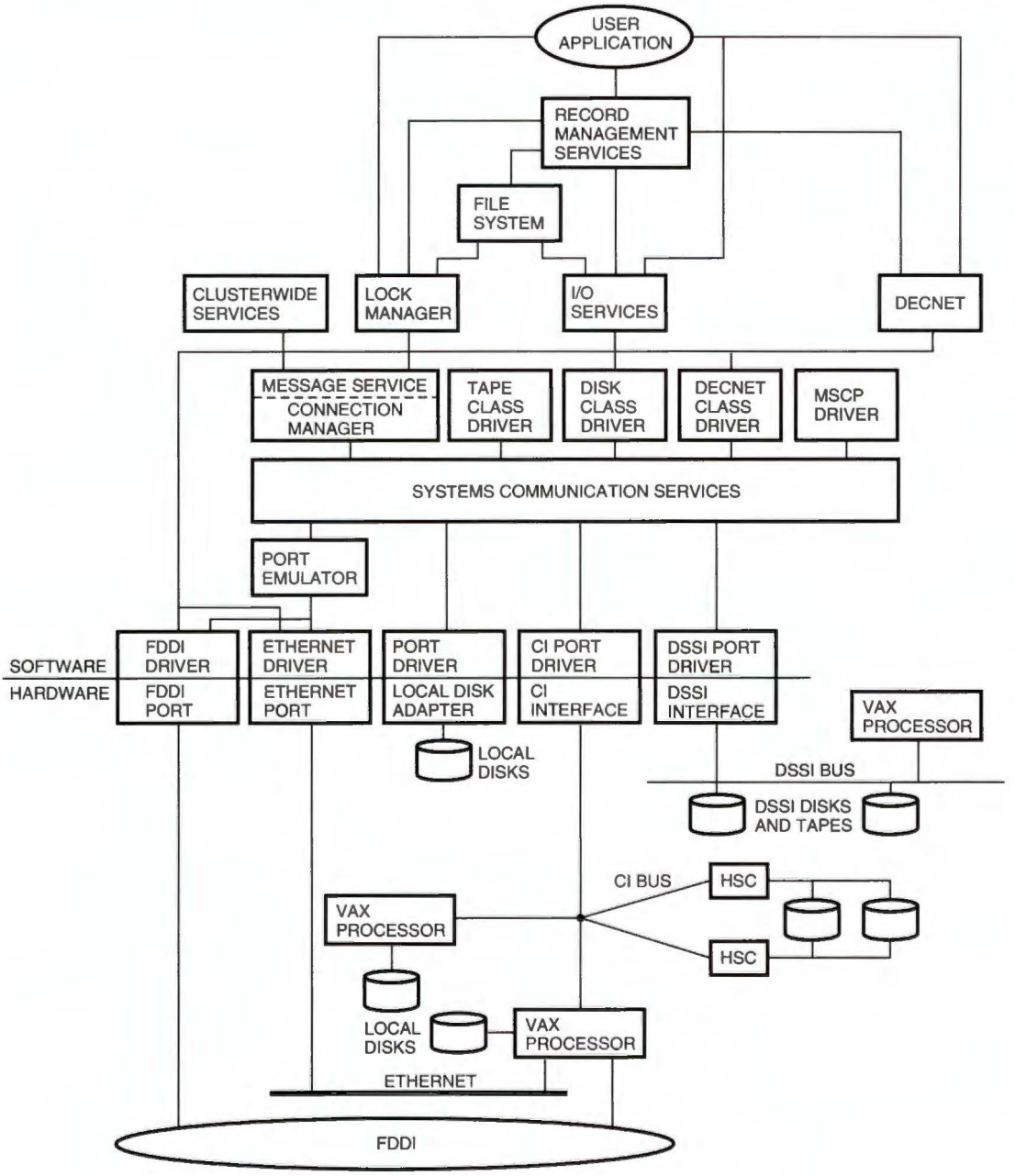


Figure 1 VAXcluster System Architecture

System Communications Services Layer

The system communications services (SCS) layer of the VAXcluster architecture is implemented in a combination of hardware and software or software only, depending upon the type of port. The SCS layer manages a logical path between each pair of nodes in the VAXcluster system. This logical path consists of a virtual circuit (VC) between each pair of SCS ports and a set of SCS connections that are multiplexed on that virtual circuit. The SCS provides basic connection management and communication services in the form of datagrams, messages, and block transfers over each logical path.

The datagram is a best-effort delivery service which offers no guarantees regarding loss, duplication, or ordering of datagrams packets. This service requires no connection between the communicating nodes. In general, the VAXcluster software makes minimal use of the datagram service.

The message and block transfer services take place over an SCS connection. Consumers of SCS services communicate with their counterparts on remote nodes using these connections. Multiple connections are multiplexed on the logical path provided between each pair of nodes in the VAXcluster system.

The message service is reliable and guarantees that there will be no loss, duplication, or permutation of message sequence on a given connection. The connection will break rather than allow the consumer of the service to perceive such errors.

The block transfer service provides a way to transfer quantities of data directly from the memory of one node to that of another. For CI ports, the port hardware accomplishes the block transfer, thus freeing the host processor to perform other tasks. Some DSSI ports use hardware to copy data and others rely on software to perform this function. Depending on the exact model of an Ethernet or FDDI port, the port software, rather than the hardware, moves the data.

System Applications

The next higher layer in the VAXcluster architecture consists of multiple system applications (SYSAPs). These applications provide, for example, access to disks and tapes and cluster membership control. The following sections describe some SYSAPs.

Connection Manager The connection manager serves three major functions. First, the connection manager knows which processor nodes are active members of the VAXcluster system and which are not. This is accomplished through a concept of cluster "membership." Nodes are explicitly added to and removed from the active set of nodes by a distributed software algorithm. In a VAXcluster system, every processor node must have an open SCS connection to all other processor nodes. Once a booting node establishes connections to all other nodes currently in the VAXcluster system, this node can request admission to the system. When one node is no longer able to communicate with another node, one of the two nodes must be removed from the VAXcluster system.

In a VAXcluster system, all nodes have a consistent view of the cluster membership in the presence of permanent and temporary communication failures. This consistency is accomplished by using a two-phase commit protocol to form the cluster, add new nodes, and remove failed nodes.

The second function provided by the connection manager is an extension of the SCS message service. This extension guarantees that the service will (1) deliver a message to a remote node or (2) remove either the sending node or the receiving node from the cluster. The strong notion of cluster membership provided by the connection manager makes this guarantee possible. The service attempts to deliver the queued messages to remote nodes. If a connection breaks, the service attempts to reestablish communication to the remote node and resend the message. After a period of time specified by the system manager, the service declares the connection irrevocably broken and removes either the sending or the receiving node from the VAXcluster membership. Thus, the service hides all temporary communication failures from its client.

This message service allows users to construct efficient protocols that do not require acknowledgment of messages. The service proved to be a very powerful tool in the design of the VMS lock manager. The delivery guarantees inherent in the service minimize the number of messages required to perform any given locking function, resulting in a corresponding increase in performance. The ability to hide failures by updating cluster membership further simplified the lock manager design and increased performance; this capability enabled the removal of logic used to handle changes in VAXcluster configurations and communication errors from all main lock manager code paths.

The third function of the connection manager is to prevent partitioning of the possible cluster

members. Partitioning of a system exists when separate processing elements function independently. If a system allows data sharing, completely independent processing can result in uncoordinated access to shared resources and lead to data corruption.

In a VAXcluster system, processors communicate and coordinate access to resources by means of a voting algorithm. The system manager assigns a number of votes to each processor node based on the importance of that node. The system manager also informs each node of the total number of possible votes. The algorithm requires that more than half of these votes be present in a VAXcluster system for nodes to function. When the sum of all votes contributed by the members of a VAXcluster system falls below this quorum, the VMS software blocks I/O to mounted devices and prevents the scheduling of processes. As nodes join the cluster, votes are added. Activity resumes once a quorum is reached.

In practice, the connection manager uses two measurements of the number of votes: static and dynamic. The static count of votes is the globally agreed on number of votes contributed by cluster members. This count is created ignoring the state of connections between nodes. The value of the static quorum changes only at the completion of two-phase commit operations, which accomplish a user-requested quorum adjustment in addition to performing the other activities mentioned earlier in this Connection Manager section.

Each node independently maintains the dynamic count. This count represents the sum of all votes contributed by VAXcluster members with which the tallying node has a functional connection. Changes in the dynamic quorum, and not the static quorum, initiate the blockage of process and I/O activity.

To provide configurations with a small number of nodes, e.g., two nodes, the concept of a quorum disk was invented. The system manager assigns a disk to contribute votes to the cluster. A node must be able to access a file on the disk in order to include the votes assigned to that disk in the node's own total. Consequently, a special algorithm is used to access the file. This algorithm ensures that two unrelated nodes cannot both count the quorum disk votes. Doing so could result in partitioned operation.

Mass Storage Control Protocol Server The Mass Storage Control Protocol (MSCP) server allows disks that are attached to one or more VAX processors to be accessed by other processors in the VAXcluster system. Thus, a VAXcluster processor may emulate a multihost disk controller by accepting and processing I/O requests from other nodes and accessing the disk indicated by the request. The server can process multiple commands simultaneously and also performs fragmentation of commands if there is not enough system buffer space to accommodate the entire amount of data at one time.

Hierarchical Storage Controllers, Local Controllers, and RF-series Integrated Storage Elements Hierarchical storage controller (HSC) servers are specialized devices that perform MSCP serving of RA-series disk drives and TA-series tape drives in a VAXcluster system. HSC servers connect directly to the CI bus. In addition to providing the host with access to the storage media, HSC servers accomplish performance optimizations such as seek-ordering and request fragmentation based on real-time head position information. The local disk controllers attached to the RA- and TA-series storage devices perform the same function for a single host processor. The RF-series integrated storage elements (ISEs) attach to a DSSI bus. Each of these disk storage devices performs its own command queuing and optimization without using a dedicated controller.

Disk Class Driver The disk class driver allows access to disks served by an MSCP server, an HSC controller, a local Digital Storage Architecture (DSA) controller, or attached to a DSSI bus. This driver provides a command queuing function that allows a disk controller to have multiple outstanding commands which can be used to provide seek, rotation, and other performance optimizations. To handle temporary communication interruptions, the driver restarts commands as needed.

VAXcluster systems can be configured so that all disks are accessed by way of redundant paths for increased availability. The way in which this is accomplished depends on the type of disk and the disk controller.

RF-series disks contain integrated controllers that connect to a single DSSI storage bus. This bus can be accessed by up to two VAX processors. Each VAX processor can then serve the disks to all other nodes in the VAXcluster system. Thus, two paths are provided to each disk.

RA-series disks connect to up to two storage controllers. These controllers can be either (1) local adapters attached directly to a single processor node or (2) HSC controllers located on the CI bus. Disks connected to local adapters can be served to other nodes of the VAXcluster system. Disks located on an HSC controller can be directly accessed by processors that are not on that bus. Thus, the use of multiple controllers when combined with disk serving provides at least two paths to a disk from every node in the VAXcluster system.

Since many paths exist to gain access to a disk, the disk class driver chooses which path to use when a disk is initially mounted by a node. If the path to the disk becomes inoperative, the disk class driver locates another path and begins to use it. Server load and type of path, i.e., local or remote, are considered when selecting the new path. This reconfiguration is totally transparent to the end user of the disk I/O service.

Tape Class Driver The tape class driver performs functions in a VAXcluster system similar to those of the disk class driver by providing access to tapes located on HSC controllers, local controllers, and DSSI buses.

VMS Components Layered on Top of SYSAPs

The SYSAPs provide basic services that other VMS components use to provide a wide range of VAXcluster features.

Volume Shadowing The volume shadowing product allows multiple disks to be utilized as a single, highly available disk. Volume shadowing provides transparent access to the data in the event of disk media or controller failures, media degradation, and communication failures.[2] The shadowing layer works in conjunction with the disk class driver to accomplish this task. With the advent of VMS Volume Shadowing Phase II, disk shadowing is extended to many new configurations.

Lock Manager The VMS lock manager is a system service that provides a distributed synchronization function used by many components of the VMS operating system, including volume shadowing, the file system, VAX RMS software, and the batch/print system. Application programs can also use the lock manager directly.

The lock manager provides a name space that is truly clusterwide. Cooperating processes can request locks on a specific resource name. The lock manager either grants or denies these requests. Processes can also queue requests. The lock manager services allow processes to coordinate the means of access to physical resources or simply provide a communication pathway between processes. Processes can use the service for such tasks as mutual exclusion, event notification, and server failure detection.[2,7] The lock manager uses the communication service provided by the connection manager to minimize the message count for a given operation and to simplify the design by eliminating the need to consider changes in cluster membership from all main paths of operation.

Process Control Services The VMS process control system services take advantage of VAXcluster systems. Applications can use these services to alter process states on remote nodes and to collect information about those processes. In the future, it is likely that other services will be extended to make optimal use of VAXcluster capabilities.

File System The VMS file system (XQP) allows disk devices to be accessed by multiple nodes in a VAXcluster system. The file system uses the lock manager to coordinate disk space allocation, buffer caches, modification of file headers, and changes to the directory structure.[8]

Record Management Services The VAX RMS software allows the sharing of file data by processes running on the same or multiple nodes. The software uses the lock manager to coordinate access to files, to record data within files, and to global buffers.

Batch/Print System The batch/print system allows users to submit batch or print jobs on one node and run them on another. This system provides a form of load distribution, i.e., generic batch queues can feed executor queues on each node. Jobs running on a failed node can be restarted automatically on another node in the VAXcluster system.

An Application Constructed Using VAXcluster Mechanisms

The VMS software build process is an example of how these mechanisms can be used to benefit application design. The VMS software build is

broken down into various phases such as fetch sources, compile, and link. The phases must execute in a given order but are otherwise independent. Each phase can be restarted from the beginning if there is an error. Each major component of the VMS operating system is processed separately during each of the phases. All sources reside on a shared disk to which all nodes of the VAXcluster system have access; the output disk is shared by all nodes also. A master data file describes the phases and the components. For a given phase, the actions required for each component are fed into a generic batch queue. This queue feeds the jobs into work queues on multiple nodes, resulting in the execution of many jobs in parallel. When all jobs of a phase have completed, the next phase starts. If a node fails during the execution of a job, that job is restarted automatically on another node either from the beginning or from a checkpoint in the job. This use of shared disks and batch queues provides great parallelism and reliability in the VMS build process.

The Impact of VAXcluster Design and Implementation on Applications

This section discusses how multiple communication paths, membership changes, disk location and availability, controller selection, disk and tape path changes, and disk failure impact application design.

Multiple Communication Paths

VAXcluster software components are able to take advantage of multiple communication paths between nodes. For greatest availability, there should be at least two physical paths between each pair of nodes in a VAXcluster system.[6]

Membership Changes

VAXcluster membership changes involve several distinct phases with slight variations depending upon whether a node is being added or removed. Adding a node to a VAXcluster system is the simplest case because it involves reconfiguration. There is a further simplification in that nodes are only added one at a time. A booting node petitions a member of an existing cluster for membership. This member then describes the booting node to all other member nodes and vice versa. In this way, it is determined that the booting node is in communication with all members of the cluster. The connection manager then adds the new node to the cluster using a two-phase commit protocol to ensure a consistent membership view from all nodes.

Removing a node is more complicated because both failure detection and reconfiguration must take place. In many cases, there may be multiple simultaneous failures of nodes and communication paths. The view of what nodes are members and which paths are functional may be very different from each node. Additionally, new failures may occur while the cluster is being reconfigured.

The initial phase involves the detection of a node failure. A node may cease processing, but other cluster members may not be aware of this fact. The communication components generally exchange messages periodically to determine whether other nodes are functioning. The first indication of a failure may be the lack of response to these messages. However, a minimum period of time must elapse before the connection is declared inoperative. This set delay prevents breaking connections when the network or remote system is unable to respond due to a heavy load. Once the communication failure is detected, the connection manager is notified by the SCS communication layer. The connection manager attempts to restore the connection for a time interval defined by the system manager using a system control parameter known as RECNXINTERVAL. Once this interval has expired, the connection and hence the remote node is declared inoperative. The connection manager then begins a reconfiguration.

Multiple nodes may attempt the reconfiguration at the same time. A distributed election algorithm is used to select a node to propose the new configuration. The elected node proposes to all other nodes that it can communicate with a new cluster configuration that consists of the "best" set of nodes that have connections between each other. "Best" is determined by the greatest number of possible votes. If multiple configurations are possible with the same number of votes, the configuration with the most nodes is selected.

Any node that receives the proposal and can describe a better cluster rejects the proposal. The proposing node then withdraws the proposal and the election process begins again. This cycle continues until all nodes accept the proposal. The cluster membership is then altered using a two-phase commit protocol, removing nodes as required.

Even when one considers the worst case of a continual failure situation, convergence on a solution is guaranteed because the connection manager does not add new nodes during a reconfiguration

and connections that fail are never used again. Thus, conditions cannot oscillate between good and bad during the reconfiguration because of nodes rebooting or because failed connections are restored. Conditions can only get worse, i.e., simpler, until failures cease to happen long enough for the reconfiguration to complete.

However, this worst-case condition is atypical; most reconfigurations are very simple. A node that is removed, as a result of a planned shutdown or because it fails, attempts to send a "last gasp" datagram to all VAXcluster members. This datagram indicates that the node is about to cease functioning. The delay present during the failure detection phase is bypassed completely, and the connection manager configures a new VAXcluster system in considerably less than one second.

Normally, the impact on an application of a node joining a VAXcluster system is minimal. For some configurations, there is no blockage of locking. In other cases, the distributed directory portion of the lock database must be rebuilt. This process may block locking for up to a small number of seconds, depending on the number of nodes, number of directory entries, and type of communication buses in use.

Application delays can result when an improperly dismounted disk is mounted by a booting node. Failure to properly dismount the disk, e.g., because of a node failure, results in the temporary loss of some preallocated resources such as disk blocks and header blocks. An application can recover these resources when the disk is mounted, but the I/O is blocked to the disk during the mounting operation. This I/O blocking has a potentially detrimental impact on applications that are attempting to allocate space on the disk. The answer to this problem is to mount disks so that the recovery of the preallocated resources is deferred. For all disks except the system disk, disk mounting is accomplished with the MOUNT/NOREBUILD command. Because a system disk is implicitly mounting during a system boot, the system parameter ACP_REBLDSYSD must be set to the value 0 to defer rebuilds. The application can recover the resources at a more opportune time by issuing a SET VOLUME/REBUILD command.

The impact on a VAXcluster system of removing a node varies depending on what resources the application needs. During the failure detection phase, messages to a failed node may be queued pending discovery that there actually is a failure. If the application needs a response based on one of these messages, the application is blocked. Otherwise, the failure does not affect the application. Once the reconfiguration starts, locking is blocked. An application using the lock manager may experience a delay, but as long as there are sufficient votes present in the cluster to constitute a quorum, the I/O is not blocked during the reconfiguration. If the number of votes drops below a quorum, I/O and process activity are blocked to prevent partitioning and possible data corruption.

Another aspect of node removal is the need to ensure that all I/O requests initiated by the removed node complete prior to the initiation of new I/O requests to the same disks. To enhance disk performance, many disk controllers can reduce head movements by altering the order of simultaneously outstanding commands. This command reordering is not a problem during normal operation; applications initiating I/O requests coordinate with each other using the lock manager, for instance, so that multiple writes, or multiple reads and writes, to the same disk location are never outstanding at the same time. However, when a node fails, all locks held by processes running on that node are released. Releasing these locks allows the granting of locks that are waiting and the initiation of new I/O requests. If new locks are granted, a disk controller may move the new I/O requests (issued under the new locks) in front of old I/O requests. To prevent this reordering, a special MSCP command is issued by the connection manager to each disk before new locks are granted. This command creates a barrier for each disk that ensures that all old commands complete prior to the initiation of new commands.

Physical Location and Availability of Disks

The application designer does not generally have to be concerned with the physical location of a disk in a VAXcluster system. Disks located on HSC storage controllers are directly available to VAX processors on the same CI bus. These disks can then be MSCP-served to any VAX processor that is not connected to that bus. Similarly, disks accessed by way of a local disk controller on a VAX processor can be MSCP-served to all other nodes. This flexibility allows an application to access a disk regardless of physical location. The only differences that the application can detect are varying transfer rates and latencies, which depend on the exact path to the disk and the type of controllers involved.

To provide the best application availability, the following guidelines should be considered:

1. VMS Volume Shadowing Phase II should be used to shadow disks, thus allowing operations to continue transparently in the event that a single disk fails.
2. Multiple paths should exist to any given disk. A disk should be dual-pathed between multiple controllers. Dual pathing allows the disk to survive controller failures.
3. Members of the same shadow set should be connected to different controllers or buses as determined by the type of disk.
4. Multiple servers should be used whenever serving disks to a cluster in order to provide continued disk access in the event of a server failure.

Selection of Controllers

Using static load balancing, the VMS software attempts to select the optimal MSCP server for a disk unit when that unit is initially brought on line. The load information provided by the MSCP server is considered in this decision. The HSC controllers do not participate in this algorithm. In addition, the VMS software selects a local controller in preference to a remote MSCP server, where possible. If a remote server is in use and the disk becomes available by way of a local controller, the software begins to access the disk though the local controller. This feature is know as local fail-back.

An advanced development effort in the VMS operating system is demonstrating the viability of dynamic load balancing across MSCP servers. Load balancing considers server loading dynamically and moves disk paths between servers to balance the load among the servers.

Disk and Tape Path Changes

Path failures are initially detected by the low-level communication software, i.e., the SCS or port layers. The communications software then notifies the disk or tape class driver of the failure. The driver then transparently blocks the initiation of new I/O requests to the device, prepares to restart outstanding I/O operations, and begins a search for a new path to the device. Static load balancing information is considered when attempting to find a new path. The path search is accomplished by sending an MSCP GET UNIT STATUS command to any known disk controller or MSCP server capable of serving the device. Some consideration is given to selecting the optimal controller; for example, the driver interrogates local controllers before remote controllers.

Once a new path is discovered or the old path reestablished, the VMS system checks the volume label to ensure that the disk or tape volume has not been changed on the device. This verification prevents data corruption in the event that someone substitutes the storage medium without dismounting and remounting the device. After a successful check, the software restarts incomplete I/O requests and allows stalled I/O requests to proceed. In the case of tapes, the tape must be repositioned to the correct location before restarting I/O requests.

If the label check determines that the original medium is no longer on the disk or tape unit, then I/O requests continue to be stalled and a message is sent to the operator requesting manual intervention to correct the problem. Attempts to reestablish the correct operation of a disk or tape continue for an interval determined by the system parameter MVTIMOUT (mount verification timeout). Once the time-out period expires, further attempts to restore are abandoned and pending requests are returned to the application with an error status. Thus, the software handles temporary disk path failures in such a transparent fashion that the application program, e.g., the user application, VAX RMS software, or the VMS file system, is unaware that an interruption occurred.

Disk Failures

If a disk fails completely when VMS Volume Shadowing Phase II software is used, the software removes the failed disk from the shadow set and satisfies all further I/O requests using a surviving disk. If a block of data cannot be recovered from a disk in a shadow set, the software recovers the data from the corresponding block on another disk, returns the data to the user, and places the data on the bad disk so that subsequent reads will obtain the good data.[2]

Summary

VAXcluster systems continue to provide a unique base for building highly available distributed systems that span a wide range of configurations and usages. In addition, VAXcluster computer systems can grow with an organization. The availability, flexibility, and growth potential of VAXcluster systems result from the ability to add or remove storage and processing components without affecting normal operations.

References

1. N. Kronenberg, H. Levy, and W. Strecker, "VAXclusters: A Closely-coupled Distributed System," *ACM Transactions on Computer Systems,* vol. 4, no. 2 (May 1986): 130–146.
2. S. Davis, "Design of VMS Volume Shadowing Phase II—Host-based Shadowing," *Digital Technical Journal,* vol. 3, no. 3 (Summer 1991, this issue): 7–15.
3. W. Snaman, Jr. and D. Thiel, "The VAX/VMS Distributed Lock Manager," *Digital Technical Journal,* no. 5 (September 1987): 29–44.
4. W. Laing, J. Johnson, and R. Landau, "Transaction Management Support in the VMS Operating System Kernel," *Digital Technical Journal,* vol. 3. no. 1 (Winter 1991): 33–44.
5. *Guidelines for VAXcluster System Configurations* (Maynard: Digital Equipment Corporation, Order No. EK-VAXCS-CG-004, 1990).
6. L. Leahy, "New Availability Features of Local Area VAXcluster Systems," *Digital Technical Journal,* vol. 3, no. 3 (Summer 1991, this issue): 27–35.
7. T. K. Rengarajan, P. Spiro, W. Wright, "High Availability Mechanisms of VAX DBMS Software," *Digital Technical Journal,* no. 8 (February 1989): 88–98.
8. A. Goldstein, "The Design and Implementation of a Distributed File System," *Digital Technical Journal,* no. 5 (September 1987): 45–55.

Lee Leahy

New Availability Features of Local Area VAXcluster Systems

VMS version 5.4-3 increases the availability of local area VAXcluster (LAVc) configurations by allowing the use of multiple local area network (LAN) adapters in the VAXcluster system. Availability is increased by enabling fail-over between LAN adapters, reducing channel failure detection time, and providing better network troubleshooting. Combined, these changes significantly increase the availability of LAN-based VAXcluster configurations by allowing the VAXcluster system to tolerate and work around network failures.

This paper describes the availability features added to local area VAXcluster (LAVc) support in VMS version 5.4-3. These features support multiple local area network (LAN) adapters, reduce the time required to detect network path (channel) failures, and provide additional support for network troubleshooting. (Table 1 presents definitions for terms used throughout the paper.)

We begin the paper with an overview of the added LAVc availability features of VMS version 5.4-3. We then present the multiple-adapter support features of the new release, with comparisons to the previous single-adapter implementation. The detection of network delays is discussed, along with how the system selects alternate paths around these delays after detection. Finally, we discuss the analysis of network failures and the physical descriptions needed to achieve the proper level of failure reporting.

Added Availability Features

VMS version 5.4-3 supports LAVc use of up to four LAN adapters for each VAX system. Availability and performance are increased by connecting each LAN adapter to a different LAN segment. Maximum availability is achieved by redundantly bridging these LAN segments together to form a single extended LAN. This configuration maximizes availability and reduces single points of failure by increasing the number of possible network paths between the different systems within the VAXcluster system.

Availability has also been increased at the applications level by reducing the time required to detect channel failures. The LAVc protocol (NISCA) sends sequenced datagrams to the remote system. If not acknowledged within 2 seconds, a datagram is retransmitted. Retransmission continues until the connection between the two systems is declared broken. However, applications can be stalled during this error recovery process. Therefore, reducing the time for detecting channel failures and retransmitting datagrams reduces the amount of application delay introduced by network problems.

VMS version 5.4-3 also increases availability by reducing the delays introduced by network congestion. This latest release measures the network delays on a channel basis. The channel with the lowest computed network delay value is used to communicate with the remote system.

LAVc network failure analysis is a new feature in VMS version 5.4-3. This feature provides an analysis of failing channels by isolating the common network components responsible for the channel failures. LAVc network failure analysis increases availability by reducing the downtime caused by failing network components. To enable this feature, the system or network manager must provide an accurate physical description of the network used for LAVc communications.

Multiple-adapter Support

This section describes the availability features added with the multiple-adapter LAVc support in VMS version 5.4-3. Some limitations of the single-adapter implementation are presented for comparison.

Single Points of Failure

In single-adapter LAVc satellites, the Ethernet adapter remains as a single point of failure. This failure "point" actually extends through the network

Table 1 LAVc Terminology

Channel	A data structure in PEDRIVER that represents a network path (see network path below). Each channel is associated with a single virtual circuit (VC).
Datagram	A message that is requested to be sent by the client of the LAN driver. A datagram does not have guaranteed delivery to the remote system. The datagram may never be sent, or could be lost during transmission and never received.
LAN Adapter	An Ethernet or fiber distributed data interface (FDDI) adapter. Each type of LAN adapter has a unique set of attributes, such as the receive ring size.
LAN Address	The network address used to reference a specific LAN adapter connected to the Ethernet or FDDI. This address is displayed as six hexadecimal bytes separated by dashes, e.g., 08-00-2B-12-34-56.
LAN Segment	An Ethernet segment or FDDI ring. Each type of LAN has a unique set of attributes, e.g., maximum packet size. LAN segments can be connected together with bridges to form a single extended LAN. However, in such a LAN, the LAN segments can have different characteristics (e.g., different packet sizes for an FDDI ring bridged to an Ethernet).
Network Path	The pieces of the physical network traversed when a datagram is sent from one LAN address to another LAN address. The network path is represented by a pair of LAN addresses, one for the local system and one on the remote system. Each network path has a specific set of attributes, which are a combination of the attributes of the local LAN adapter, the remote LAN adapter, and each of the LAN segments and LAN devices on the path between them.
PEDRIVER	The VMS port driver that provides reliable cluster communication utilizing the Ethernet.
Virtual Circuit	A data structure in PEDRIVER that represents the data path between the local system and the remote system. This data path provides guaranteed delivery for the messages sent. PEDRIVER's datagram service, along with an error recovery mechanism, ensures that a message is delivered to the remote system or is returned to the client with an error. A virtual circuit (VC) has one channel for each network path to the remote system.

components common to all of the network paths in use for cluster communication. The combination of VMS version 5.4-3 with multiple LAN adapters removes the LAN adapter as a single point of failure in the local system. Additionally, the use of multiple LAN adapters connected to an extended LAN creates multiple network paths to remote systems. This configuration results in a higher tolerance for network component failures and higher cluster availability.

Adapter Selection

The single-adapter implementation is configuration-dependent and does not allow the system manager a choice of adapters. The multiple-adapter support in VMS version 5.4-3 configures the system for maximum availability by starting the LAVc protocol on all LAN adapters in the system. Support is also provided to start and stop the LAVc protocol on the LAN adapters. This support allows the system manager to select which LAN adapters will run the LAVc protocol.

The means of locating the LAN devices in the system has also changed. The system now maintains a list of LAN devices. As each LAN device driver is loaded into the system, an entry is appended to this list. A new support routine steps through this list and returns a pointer to the next LAN device in the system. The single-adapter implementation requires code changes in PEDRIVER to add a new LAN device; the new implementation no longer requires these changes.

Channel Control Handshake

The channel control handshake is a three-way message exchange. The exchange starts when a HELLO message is received from a remote system and the channel is in the closed state, or any time a CCSTART message is received. Upon receiving a HELLO message on a closed channel, the system responds with a CCSTART message.

Upon receiving a CCSTART message, the system closes the channel if the PATH bit was set. In all cases, if the cluster password is correct, the system responds with a VERF message. Upon receiving the VERF message, the remote system verifies the cluster password. If the password is correct, the remote system sends an acknowledgment (VACK) message and marks the channel as usable by setting the PATH bit. The local system, upon receiving the VACK message, also marks the channel as usable by setting the PATH bit.

The channel control handshake now verifies the network path used by this channel, instead of verifying the virtual circuit (VC) as in the single-adapter implementation. Additionally, the handshake is used to negotiate some parameters between the local and remote systems on a channel basis (instead of assuming that the parameters are common for all channels connected to the VC).

Packet size and pipe quota are two characteristics that are now arbitrated between the two systems. These parameters are negotiated on a channel-by-channel basis to allow different channels to fully utilize the capabilities of the specific network path.

With the introduction of FDDI, larger packet sizes are now supported. The channel handshake between two nodes negotiates a packet size that is supported by the entire network path. Any path that utilizes an Ethernet must use a packet size of 1498 bytes or smaller. An FDDI-to-FDDI path on the same extended ring must use a packet size of 4468 bytes or smaller. An increased packet size reduces the number of messages required when large blocks of data are sent. This increase in packet size results in fewer messages, less handshaking, and thus better network efficiency.

The PIPE_QUOTA value is used to limit the number of messages sent to the remote system before waiting for an acknowledgment. PIPE_QUOTA was implemented to help prevent receiver overrun on the remote system. Instead of using a fixed value, the new implementation uses a value specified by the LAN driver. This value factors in the LAN device's receive ring size and is typically larger than the fixed value of eight messages used previously. Increasing the PIPE_QUOTA value allows more data to be sent between the nodes before an acknowledgment message is required, thus increasing the protocol's efficiency and reducing the network traffic.

These new features in VMS version 5.4-3 have reduced the amount of handshaking required to move data and the number of messages required to move large amounts of data. The result is greater applications availability through fewer network-based delays.

Use of Hello Messages

The single-adapter implementation uses a HELLO message to maintain the VC and not the channels. Also, the handshake to verify connectivity is performed by the VC, which forces all channels to use the same characteristics. In comparison, the multiple-adapter implementation uses HELLO messages to trigger the channel handshake, test the network path and maintain the channel in the open state, and continuously verify the network topology.

To maintain the channel and test the network path, each system multicasts a HELLO message through each of its LAN adapters every 3 seconds. Upon receipt of a HELLO message (if the channel is not open), a channel handshake begins. If the channel is open, the network delay is computed and the channel packet size is verified. When an open channel does not receive a HELLO message within 8 seconds, it declares a listen time-out and the channel is closed.

Additional topology change detection is required because FDDI-to-FDDI communications use large packets. If two systems using FDDI adapters exchange channel control messages, then both can agree on a large packet size. However, if the network is configured in the dumbbell configuration, then only the small packet size can be used. (The dumbbell configuration consists of two FDDI rings separated by an Ethernet segment.)

Detection of the dumbbell configuration is performed using the priority field in the frame control byte of the FDDI message header. This field does not exist in Ethernet messages and must be created when forwarding an Ethernet message to an FDDI ring. Ethernet-to-FDDI LAN bridges set this field's value to zero. All LAVc messages transmitted by the FDDI adapters use a non-zero value for the priority field. When a channel control message is received, the value of this field is checked. If the value is non-zero, then large messages can be used because the message did not traverse an Ethernet segment.

The priority field is also verified every time a HELLO message is received and the channel is open. A topology change is detected when a change in the priority value is received. If the priority value goes from zero to non-zero, the packet size is renegotiated and a larger packet size may be used. If the priority value goes from non-zero to zero, the channel packet size must be reduced. If this is the only channel with a larger packet size, then the VC closes and forces the two systems to reassign the message sequence numbers.

Listen Time-out

VMS version 5.4-3 now consistently times out channels in 8 to 9 seconds, whereas the single-adapter implementation detects the failure in 8 to 15 seconds. Reducing this time reduces the delays experienced by applications when a LAVc node is removed from the cluster. The result is an increase in applications availability.

The single-adapter implementation traverses the VC list and scans each of the receive channels (RCH structures embedded in the VC) to check for time-out. Because this scan is CPU-intensive, the algorithm was designed to scan the VC list only once every 8 seconds. Reducing this scan time required the design of a new algorithm that reduces the CPU utilization required to locate the channels that have timed out.

The VMS version 5.4-3 implementation places each open channel into a ring of time-out queues. The time-out routine maintains a pointer into the ring of queues corresponding to the 8-second time-out. Each second, the time-out routine executes, removes any channels pointed to by the time-out pointer, and calls the listen time-out routine for the channel. Next, the time-out pointer and the 8-second time-out pointer are updated to point to a new set of queue headers in the ring. Active channels and channels receiving HELLO messages are inserted into the ring of queues pointed to by the current time pointer, which prevents them from timing out. This implementation reduces CPU utilization during the time-out scan by looking at only the channels that have timed out.

Changes to Virtual Circuit Maintenance

The single-adapter implementation closes the VC and performs a channel control handshake every time a new channel is established. This implementation also forces each channel to use the same characteristics, specifically packet size, thereby reducing the characteristics to the lowest common denominator.

VMS version 5.4-3 does not close the VC each time a new channel is established. The channel handshake affects only the channel and is used to negotiate the channel characteristics, including packet size. The VC remains open as long as a channel with the corresponding packet size is open. This maintenance increases applications availability by allowing channels to fail and reestablish transparently without disrupting service at the VC and systems communication services (SCS) layers.

One Channel with Matching Characteristics Required The VC can be opened as soon as the first channel to the remote system is opened. When the VC opens, its packet size is set to the packet size of the channel being used. The VC can remain open as long as at least one channel with a compatible packet size is open. The packet size is compatible if the channel packet size is greater than or equal to the packet size currently in use by the VC.

Transfers restricted to an FDDI ring can use a larger packet size than those that traverse an Ethernet LAN segment. PEDRIVER now supports variable packet sizes up to the size supported for the FDDI ring. Each time the VC switches channels, the new channel characteristics are copied into the VC. The result is that as soon as the VC switches to using the FDDI-to-FDDI channel, it also switches to using the larger packet size.

Receive Message Caching VMS version 5.4-3 introduces a receive message cache to prevent any performance degradation when messages are received out of order. Because of transmission and network delays, messages are typically received out of order at approximately the time a channel switch occurs. Also, most of the channel selections are invoked after locating a channel with a lower network delay value, thus increasing the probability that messages will be received out of order.

Channel Failure Not Displayed The multiple-adapter implementation does not display any messages when a channel fails. This choice was made to maintain compatibility with the previous implementation. We also wished to reduce the number of console messages and still provide enough data to isolate the problem. However, without some channel failure notification, all but one channel could fail without notice, thus negating all the availability that was introduced by using multiple adapters.

The LAVc network failure analysis allows the system or network manager to select one of the following levels of channel failure notification: no notification, if not enabled; channel failure notification, when barely enabled; or isolation of the failing network component, when fully enabled. When this feature is fully enabled, a failing network component typically generates only a single console message instead of displaying tens or hundreds of channel failure messages.

Channel Selection

VMS version 5.4-3 bases its selection of a single transmit channel for a remote system first, on the packet size and second, on the network delay value. The channel selection algorithm searches for an open channel with a compatible packet size so that the VC does not have to be broken. If more than one channel has a compatible packet size, the

network delays are compared and the channel with the lowest network delay value is chosen. The selected channel is used until it fails, encounters an error, or a channel with a lower network delay value is found.

Channel selection is performed independently for each remote system. This implementation means that a two-node cluster increases its availability through the use of more LAN adapters, but does not achieve a performance benefit by increasing the number of LAN adapters above two. Larger clusters, however, can take advantage of the additional LAN adapters and thus achieve better cluster performance. Multiple LAN adapters can also increase the bandwidth available for use by the LAVc protocol. However, the actual performance is very configuration- and application-dependent.

Channel selection is limited to the transmit channel, but all channels are used to receive data. The receive cache helps prevent retransmission by the remote system by placing messages received out of order into the receive cache until the previous messages are received. This receive algorithm is compatible with any transmit channel selection algorithm, e.g., in PEDRIVER or in any component implementing NISCA.

Multiple-adapter Availability Summary

The multiple-adapter LAVc support added to VMS version 5.4-3 increases the availability of applications and of the overall cluster. Availability is increased by removing the LAN adapter as a single point of failure. Cluster availability is enhanced through continuous testing of the network paths and correction for network topology changes.

This implementation also increases network utilization and cluster performance by taking full advantage of a channel's characteristics. Larger receive ring sizes reduce the protocol handshaking overhead. Moreover, larger packet sizes reduce the number of messages that must be sent for large transfers.

The next section discusses how the PEDRIVER detects network delays and selects alternate paths.

Network Delay Detection

VMS version 5.4-3 increases application availability by detecting significant network delays and selecting alternate paths. As the network gets busy, it becomes more difficult for a LAVc node to send cluster messages. These delays in network communications cause delays in cluster traffic and translate into delays in the applications. Thus, through delay detection and the use of alternate paths, VMS version 5.4-3 reduces the delays for applications and increases overall cluster performance.

Assumptions and Delay Calculations

PEDRIVER computes network delays through a series of assumptions. The primary assumptions are that the transmit and receive delays for a path are equal, and that there are small internal delays associated with the LAN device. Although these assumptions are occasionally invalid, PEDRIVER uses them because there are no round-trip messages available in the NISCA protocol to compute the delay.

As the first step in the delay calculation for each channel between nodes, each node time-stamps the HELLO message just prior to transmission. When the HELLO message is received, the time stamp is subtracted from the local system time. This resulting value equals the sum of the transmit queue delay, the network delay, the receive queue delay, and the difference in the two system times. Applying the assumptions reduces this value to the sum of the network delay and the difference in the two system times.

The second step of the delay calculation is to compare the delay times between different channels to the same remote system. This comparison is a subtraction of the values computed above for each channel. The computation removes the common factor (the difference in the two system times) and results in the comparison of the two network delays. When multiple channels exist, PEDRIVER attempts to use the channel with the lowest network delay value.

Problems and Benefits Associated with the Assumptions

The assumptions in the network delay calculation do not always hold true. The arbitration delay to transmit a message on the Ethernet, between a pair of systems, is not always equal in both directions. Over the long term, this assumption would be valid if the systems are sending the same number of messages in each direction; however, this is not typically the case. When this assumption does not hold true, i.e., if the transmit delay is longer than the receive delay, then additional delay is introduced when transmitting messages using this channel.

The assumption that internal delays are small depends upon the network traffic and the transmit traffic generated for an adapter by the other LAN

clients. If another LAN client is a heavy user of a particular LAN adapter, then transmissions from this adapter experience additional queue delays while waiting for the adapter. If the network is busy, messages in the transmit queue have an additional wait.

Finally, the network delay computed is the delay from the remote system to the local system. Since the delay is not always symmetric, it does not always represent the delay in the other direction, i.e., transmitting messages to the remote system. Yet, because the NISCA protocol does not have any round-trip messages, this is the best possible delay value.

Even with these problems in the assumptions, the network delay calculations increase the availability of the cluster by detecting large network delays. With this data, PEDRIVER is usually able to select alternate paths around the network delays when multiple channels exist, providing better cluster performance and availability.

Figure 1 represents an example of network delay detection. If LAN segment A is very busy, then PEDRIVER can detect an additional network delay for channels A1–B1, A1–B2, and A2–B1. PEDRIVER can then select an alternate path, that is, transmit packets only on channel A2–B2. Use of channels A1–B1, A1–B2, and A2–B1 can resume when the network traffic level on LAN segment A is reduced to about the level of LAN segment B, or if channel A2–B2 fails.

LAVc Network Failure Analysis

VMS version 5.4-3 uses multiple LAN adapters to increase availability by working around network delays and failures. Channels fail as network failures occur, reducing the availability provided by these extra channels. However, the VC remains open, allowing cluster communication as long as a single channel remains open.

To maintain compatibility with previous VMS versions, only VC failures are displayed on the local console. Displaying messages about channel failures would only indicate a problem without helping to locate the cause of the failure. Also, as the cluster configuration gets larger, or the number of LAN adapters increases, channel failure messages increase (depending on what component failed) beyond the point where they are helpful. Yet to maintain cluster availability, the system or network manager needs to be told of the channel failures that are reducing the availability.

The LAVc network failure analysis, introduced with VMS version 5.4-3, is used to analyze the network failures and display the OPCOM messages that call out the failing network component. This support requires a description of the physical network used for LAVc communications. Depending upon the description supplied, the system or network manager can select the level of failure reporting. This level may range from channel failure reporting to calling out the actual component that failed.

Display of Channel Failures

There is a significant difference between displaying the channel failures and performing LAVc failure analysis. This difference is shown in Figure 2, which represents a multiple-adapter LAVc configuration.

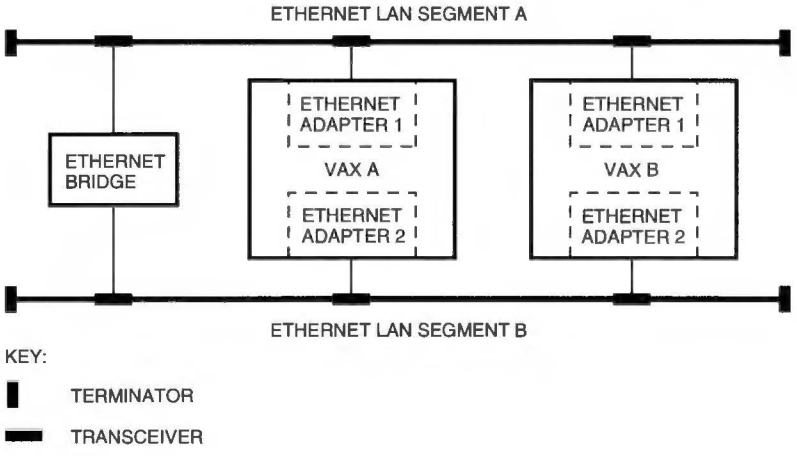


Figure 1 Network Delay Detection

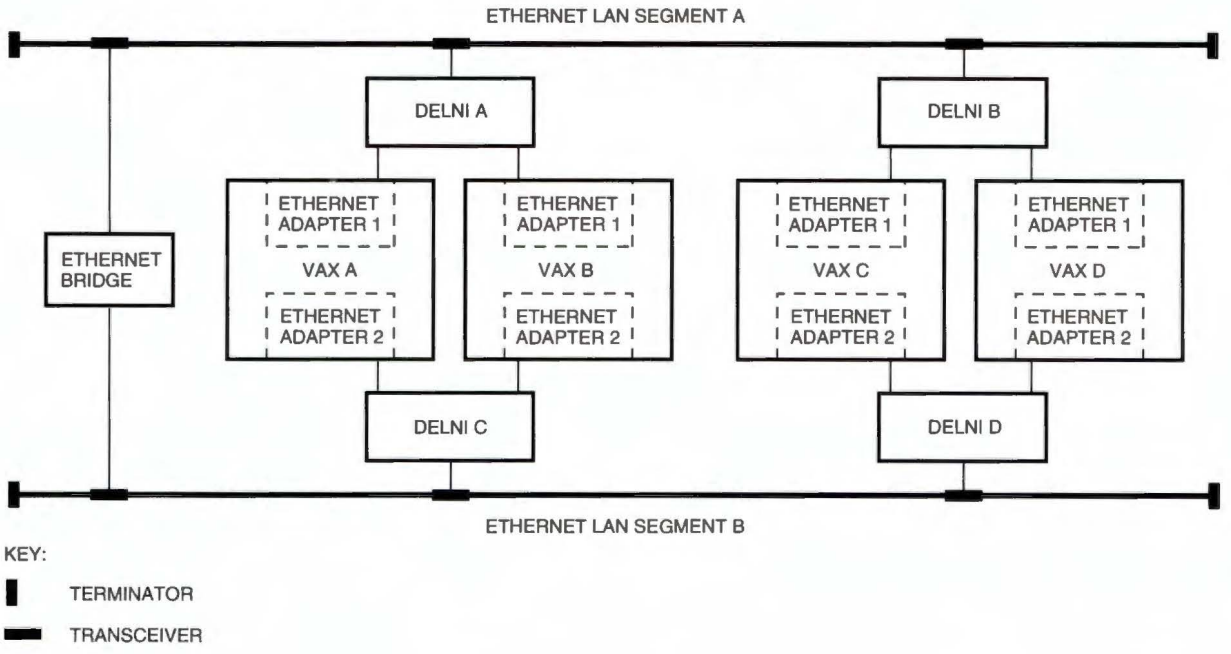


Figure 2 Multiple-adapter Channel Failure

Looking from system VAX A, the following channels exist: A1–A2, A2–A1, A1–B1, A1–B2, A2–B1, A2–B2, A1–C1, A1–C2, A2–C1, A2–C2, A1–D1, A1–D2, A2–D1, and A2–D2. Let us assume that DELNI B fails, causing the following channel failures: A1–C1, A2–C1, A1–D1, and A2–D1. A display of channel failures would show that some interesting event had just occurred but would leave it up to the system or network manager to isolate the actual failure. Also, since other channels are still open to VAX C and VAX D (A1–C2, A2–C2, A1–D2, and A2–D2), these nodes still remain in the cluster. However, the number of channels to these nodes has been halved, reducing cluster availability.

LAVc network failure analysis uses the physical network description to analyze channel failures. The working channel A1–C2 indicates that VAX A, A1, DELNI A, LAN segment A, Ethernet-to-Ethernet LAN bridge, LAN segment B, DELNI D, C2, and VAX C function. The working channel A2–D2 indicates that A2, DELNI C, D2, and VAX D also function. The remaining components are DELNI B, C1, and D1. By reviewing the failing channels for common failures, we see that two channels use component C1, two channels use component D1, and all four channels use component DELNI B. Therefore, DELNI B has the highest probability of causing the failure and is the only network component displayed on the console.

In this small cluster configuration, LAVc network failure analysis has reduced the messages displayed, i.e., from four channel failure messages to one component failure message. This simpler display provides timely notification and better isolation of network component failures, allowing the system or network manager to repair the network earlier and restore the full availability of the cluster.

Physical Network Description

LAVc network failure analysis requires a description of the physical network. This description lists the components used by the LAVc and the network paths that correspond to the LAVc channels.

The network component description consists of several pieces of data, including a component type and text description provided by the system or network manager. Some component types will require additional data. There are several types of network components: NODE, ADAPTER, COMPONENT, and CLOUD. Each NODE component requires a unique node name associated with it that matches the SCSNODE SYSGEN parameter. The ADAPTER component has at least one and sometimes two LAN addresses associated with it. One LAN address is the hardware address and the other, when specified, is the DECnet LAN address. COMPONENTs are used to describe all pieces of the network, both working

and nonworking. CLOUDs describe portions of the network that are working only if all paths are working. Any path failure implies that the CLOUD component may not be working.

Component descriptions can range from actual devices and cables to internal CPU bus adapters. When the component is defined, an ID value is returned for use in the network path description. The choice of the components described is left to the system or network manager and allows the manager to select the desired level of network analysis. Each network component has a reference count and a working count. The reference count is incremented when a network path is defined that utilizes the network component. The working count is incremented each time a LAVc channel is opened, and decremented each time an open LAVc channel is closed.

The network path description consists of a directed list of component identifier (ID) values. For proper analysis, this list must start with the ID value for the local node. Each successive ID value in the list must be associated with the next network component through which a message would travel when using this path. The final component ID value is that of the remote node.

Each network path description must contain two node ID values and two adapter ID values. To be useful for analysis, the path description must contain the node ID value for the node running the analysis. Without this node ID value, the path cannot be matched with any of the LAVc channels on that node.

Channel Mapping and Processing

The network path descriptions are matched with the LAVc channels by using the LAN addresses. If possible, only the LAN hardware address is used for the mapping function. This mapping provides the best analysis because it remains constant with respect to any LAN adapter. In clusters running mixed VMS versions, the LAN hardware address is not available for systems running a version prior to VMS version 5.4-3. In prior versions, the DECnet LAN address is used for the mapping function.

Each time a LAVc channel is opened, the network path database is searched to locate a matching network path description. If found, this description is connected to the channel and a scan of all the components in the path is performed. For each component in the path, the working count is incremented. If the component switches from not working to working, then a WORKING message is displayed.

When a LAVc channel fails, the corresponding network path is placed on a failure list. The network path is then scanned and the working count for each component is decremented.

Failure Analysis

Related channel failures are collected by delaying 10 seconds following the channel failure. Each channel failure extends the time delay to the full 10 seconds. Once the 10-second delay has elapsed following the last channel failure, the full list of failing network paths is processed.

Computing the failure probabilities begins by reviewing each of the components in the network path. If a component cannot be proven to work, then it is placed on the suspect list and the component's suspect count is incremented. A component is working if the working count is non-zero; a CLOUD component is working if the working count equals the reference count. This step ends with a list of suspect components, each with a suspect count that represents the number of times this component could have caused the failure.

Suspects are selected by comparing the suspect counts for each of the components in a network path. Each network path is reviewed independently and a primary suspect is selected. The primary suspect is the first component with the highest suspect count in the network path. Secondary suspects are the other components in the network path with the same suspect count value. The primary and secondary suspects are displayed after all the network paths have been reviewed. The other components in the suspect list are removed from the list, and are not displayed because the failure analysis judged them to be unrelated to any of the channel failures.

There are several limitations to the failure analysis. The analysis requires an accurate description of the physical network. The failure analysis is also looking for a common network component failure. Therefore, an incorrect analysis results from either an inaccurate network description, multiple related failures, or too much detail.

The key to a valid network failure analysis is the correct description of the physical network. In Figure 2, if the network path A1–B1 incorrectly listed DELNI B, then the failure analysis would find that DELNI B is working and remove it from the suspect list. The final analysis would list both C1 and D1 as the failing components. Validation of the network description can be performed by network

fault insertion and by reviewing the network failure analysis. If the description is accurate, then the failure analysis should display the expected messages. If an inaccurate network description exists, unexpected messages may be displayed. In such cases, the network description should be reviewed.

Multiple related failures may also cause an incorrect failure analysis. Referring again to Figure 2, assume a correct network description. Instead of a DELNI B failure, assume that both C1 and D1 have failed. The failure analysis reviews the network description and locates the single component DELNI B because it is common to all of the failures. In this case, the failure analysis does correctly locate the area of the network (something connected to DELNI B). However, further review is required to identify that DELNI B itself has not failed, but rather both C1 and D1.

The choice of the network description, the number of components defined, and the path descriptions, is left to the system or network manager. This choice allows the manager to select the level of failure reporting needed to troubleshoot the network. However, when the physical network description includes too much detail (e.g., transceiver cables), it becomes difficult for the failure analysis to reduce the components to a single failure. Instead, a primary suspect and several secondary suspects are usually displayed. Too much detail also requires more CPU cycles and memory for analysis, and in general is a bad trade-off.

In Figure 2, if the Ethernet adapter C1 fails, and the transceiver cables are listed in the network description, then the failure analysis displays two messages. The primary suspect is listed as the transceiver cable because it is the first component that matches the failure in the path from A to C. The Ethernet adapter C1 is listed as a secondary suspect, because its suspect count matches the suspect count of the primary suspect. In this example, there are no network paths described that use Ethernet adapter C1 without using the transceiver cable connected between C1 and DELNI B. With the network description provided, there is no way to distinguish between these two components. Therefore, both are displayed when either is a primary or secondary suspect.

Benefits

The LAVc network failure analysis, combined with an accurate description of the physical network, enables the system or network manager to maintain the increased availability gained with the use of multiple LAN adapters. Timely analysis and reporting of network component failures significantly reduces troubleshooting times and increases the overall cluster availability.

Summary

VMS version 5.4-3 increases the availability of Local Area VAXcluster configurations by providing the following features:

- Faster detection of channel failures
- Support for the use of multiple adapters
- Support for the use of additional network paths
- Detection of network congestion
- Analysis of network failures

The goals of these features are to

- Provide higher cluster availability
- Work around network congestion and network component failures while keeping the cluster running
- Detect problems earlier and report them more accurately, with network data that helps isolate the failing network components

In addition to meeting these goals, the features in VMS version 5.4-3 increase the cluster communication bandwidth.

Acknowledgments

I want to thank Kathy Perko and Steve Mayhew for their help with the design of the multiple-adapter version of PEDRIVER. Kathy reviewed the code during the implementation and provided valuable input for both the code and this paper. Thanks to Scott H. Davis, Sandy Snaman, and Dave Thiel for their contributions to the new PEDRIVER design. Thanks also to the LAN Group (Linda Duffell, Dave Gagne, Rod Gamache, Bill Salkewicz, and Dick Stockdale) for the VAX communication interface to the LAN drivers, which simplified the design of the new PEDRIVER. I also wish to acknowledge the LAN Group for their help during the debug phase of this implementation.

Richard E. Stockdale
Judy B. Weiss

Design of the DEC LANcontroller 400 Adapter

The DEC LANcontroller 400, Digital's XMI-to-Ethernet adapter (DEMNA), connects systems based on the Digital XMI bus to an Ethernet/IEEE 802.3 local area network (LAN). These systems use the XMI bus either as the system bus (VAX 6000 systems) or as an I/O bus (VAX 9000 systems). The new systems, which can utilize the full bandwidth of the Ethernet, are characterized by increased host processor speeds. The DEMNA adapter was designed to support these I/O requirements. In addition, console and monitor facilities were built into the adapter firmware for debugging, verification, and user visibility. The adapter's performance for small packets exceeds system capabilities, and Ethernet bandwidth is the limiting factor for large packets.

The high-performance DEC LANcontroller 400, Digital's XMI-to-Ethernet adapter (DEMNA), connects a system based on the Digital XMI bus to an Ethernet/IEEE 802.3 local area network (LAN). This adapter is intended for Digital systems that use the XMI bus either as the system bus (VAX 6000 systems) or as an I/O bus (VAX 9000 systems). It is an intelligent adapter that implements the physical layer and part of the data link layer of network protocol. The term intelligent refers to the packet processing performed by the adapter as part of the data link layer.

The DEMNA adapter was needed to support the I/O requirements of the VAX 6000 and VAX 9000 systems, which can utilize the full bandwidth of the Ethernet. The adapter also provides the ability to configure these systems without a BI bus. For these systems, the DEMNA adapter is the only Ethernet connection available.

The DEMNA adapter is controlled by a port driver that resides in host memory. The interface between the port driver and the DEMNA firmware (the port) is a ring-based design which is optimized for low system overhead and high performance.

The DEMNA adapter has the following major features:

- Supports Ethernet/IEEE 802.3 protocols
- Supports up to 64 users (each one a separate protocol such as local area transport [LAT] software, DECnet network software, or clusters)
- Supports two modes of addressing: VAX virtual addressing and 40-bit physical addressing
- Allows buffer chaining on transmit
- Performs packet filtering and validation on receive
- Supports Digital's maintenance operations protocol (MOP) functions
- Provides support for diagnostic routines and field service functions implemented through the system console or diagnostic software
- Has console and monitor facilities that allow a console user to monitor DEMNA operation and network utilization

This paper begins with a logic overview of the DEMNA device. The sections that follow discuss the factors that influenced design and implementation, describe the major performance metrics and user visibility operations, and review the design results and future needs.

Logic Overview

The DEMNA adapter is a single-board XMI adapter based on complementary metal-oxide semiconductor/transistor transistor logic (CMOS/TTL) technology. As shown in Figure 1, the hardware consists of four separate subsystems:

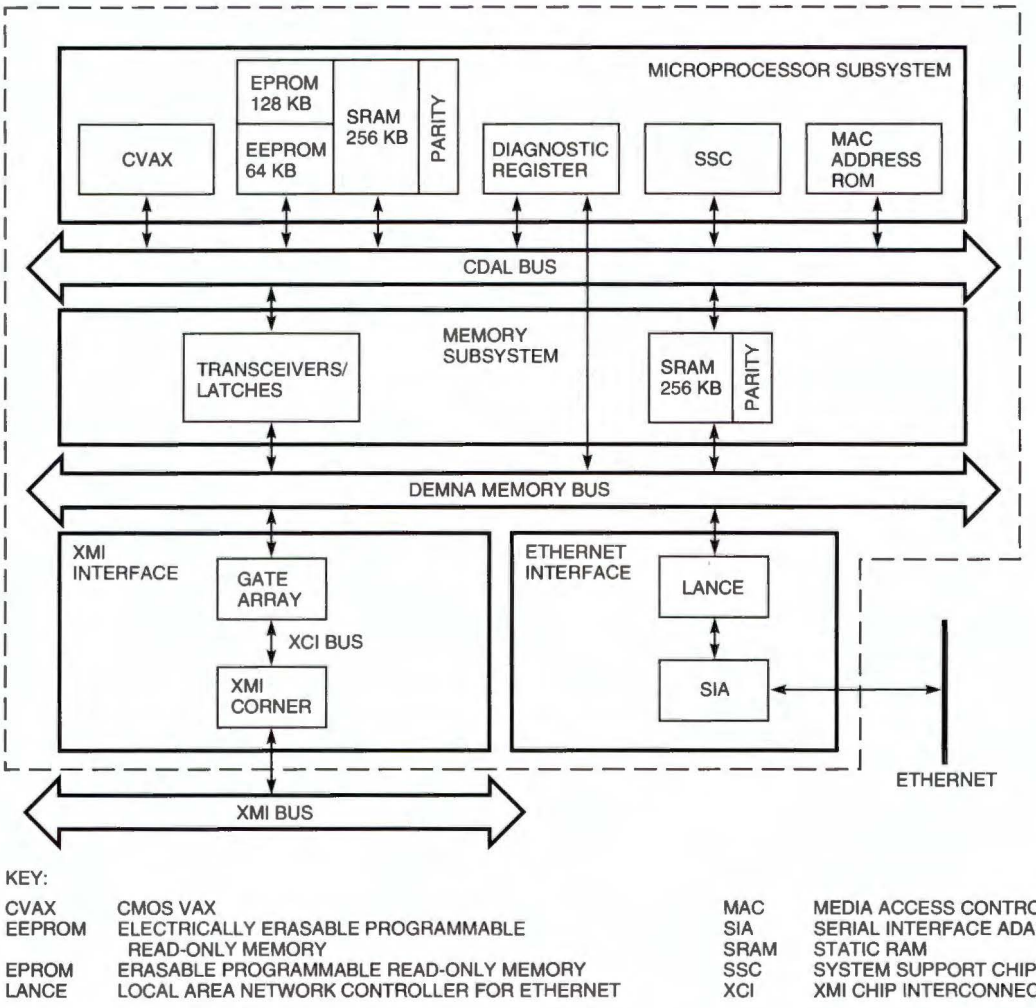


Figure 1 DEMNA Block Diagram

- Microprocessor
- Direct memory access (DMA) and shared memory
- XMI interface
- Ethernet

The microprocessor subsystem contains the CMOS VAX (CVAX) processor, system support chip (SSC), boot read-only memory (ROM), Ethernet address programmable read-only memory (PROM), electrically erasable programmable read-only memory (EEPROM), and random-access memory (RAM). The microprocessor subsystem provides an internal, high-speed CDAL bus so that the CVAX processor can fetch its instructions and execute them without being delayed by the other controllers on the module. The firmware is stored in EEPROM, but is copied to RAM for execution. The boot ROM contains the initialization code and diagnostics. This subsystem also provides a console interface through the SSC for diagnostics, module debugging, and network monitoring.

The DMA and shared memory subsystem provides the means of communication between the CVAX processor and the other subsystems. The devices arbitrating for this shared memory are the CVAX processor, the gate array, and the Local Area Network Controller for Ethernet (LANCE) chip.

The XMI interface subsystem contains the XMI network adapter (XNA) gate array and the XMI corner. The XNA gate array is the data-move engine for the DEMNA adapter and contains all the XMI-required registers.

The Ethernet subsystem contains the LANCE chip, the serial interface adapter (SIA) chip, and various bus interface logic modules. The Ethernet subsystem receives packets from the Ethernet and stores them in the shared memory. When transmitting a packet on the Ethernet, the LANCE chip gets the packets from shared memory and transmits them on the Ethernet.

Design

The design of the DEMNA adapter was influenced by many factors, including previous adapter design experiences, available hardware such as Ethernet chips, and system requirements. The DEMNA team was assigned the following tasks:

- Produce a working Ethernet adapter that could be used by operating systems such as VMS, ULTRIX, ELN, and custom operating systems on hardware configurations that use the XMI bus as a system bus or an I/O bus
- Deliver high performance, measured by the amount of Ethernet bandwidth supported at various packet sizes, with minimized host overhead
- Supply debugging features for design verification and field maintenance of the adapter

First, we reviewed previous adapters to determine what improvements could be made. We learned that a complex host interface complicated host software and adapter firmware and greatly affected performance. One of these adapters, the Digital BI Ethernet Network Adapter (DEBNA), implemented a generic port interface that used interlocked queues containing a queue entry with a buffer name that indexed into a buffer descriptor table (i.e., an additional level of indirection). In addition to the firmware complexity, the hardware was not well suited to a complex port interface.

Another area in which improvements could be made over previous Ethernet adapters was the amount of processing performed by the host processor during receive packet filtering, address translation, and buffer copies. Overall system performance improves if this processing can be reduced by performing part or all of these functions in the adapter. This difference transforms the adapter from a dumb adapter (much of the data link processing performed by the host) to an intelligent adapter (much of the processing performed by the adapter).

The results of our analysis of older Ethernet adapters led us to choose a design that employs a simple host interface, off-loads the host whenever possible, uses rings instead of queues, and supplies the address of the buffer directly with the ring entry rather than indirectly through another data structure.

The design of the adapter was now consistent with the needs of the new VAX 6000 and VAX 9000 systems. These systems, characterized by increased host processor speeds, needed increased I/O performance. The task of the DEMNA team was to fill that need for Ethernet I/O.

Type of Adapter

The DEMNA product is a store-and-forward adapter, i.e., it copies data to and from host memory by way of temporary storage on the adapter. This data transmission differs from that of a cut-through adapter in which data flows directly between host memory and the transmission medium. However, the DEMNA adapter is actually able to gain some of the benefits of cut-through on the receive side.

Host Interface

We designed a simple host interface, using rings instead of queues. Interrupts to the host were kept to a minimum, from one interrupt per packet at light loads to a fraction of that number under heavy loads. As seen in Figure 2, the port and the port driver (host) share the following data structures, which reside in host memory:

- Port data block. This structure gives the port the location of the rings and page tables in host memory and is a repository for error information.
- Command and receive rings. These rings contain information describing outstanding command and transmit requests and buffer information for receive buffers.
- Transmit, receive, and command buffers. These buffers contain packet data and command data.

These data structures constitute the primary means of communication and data transfer between the port and the port driver. Control status registers (CSRs) are provided for port poll demand registers, XMI context, and port initialization.

Two rings are used in the host interface: the command ring and the receive ring. Each ring consists of 1024 bytes of physically contiguous memory, and each ring contains entries that describe a buffer or a set of buffer segments (when chaining transmit buffers). The number of entries in the receive ring

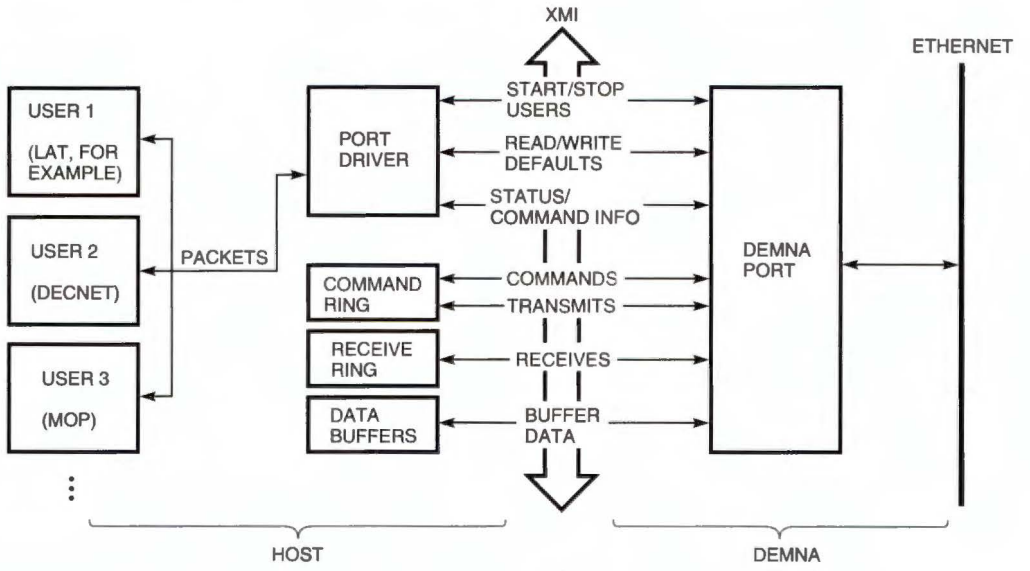


Figure 2 DEMNA Port Interface

is fixed, since each entry points to a single contiguous buffer. The size of each transmit ring entry is variable and is fixed at initialization time.

The port and port driver process the entries in each ring in sequential order, starting with the first entry. A ring entry can be processed only by its owner. When the last entry in the ring is reached, processing starts again with the first entry.

Host interrupts are minimized by using a ring release function, which counts the number of ring entries processed for completion by the port and the port driver. The port driver counts the number of completed entries and writes this count to a completion CSR when it has finished processing all the completed transmit and receive ring entries. The port maintains the same count and issues another interrupt whenever it sees that its count and the count last written by the port driver are different. This function ensures that the port driver is interrupted only when it stops processing the rings because there is nothing else to process. The port driver can process multiple completed transmits and receives after each interrupt as well. Thus, no spurious interrupts are issued and the number of interrupts is reduced by processing multiple completions at once.

Adapter Design

The firmware is written in VAX MACRO code. An alternative was to use MACRO for the transmit and receive paths and a higher-level language for initialization, shutdown, and error handling. However, this approach was not chosen because it complicates the interface and would have resulted in firmware size difficulties.

CVAX RAM (used by the CVAX processor exclusively) consists of 256 kilobytes and contains the firmware and data structures (the firmware is copied to RAM during self test). Smaller RAMs would have been slightly less expensive but would have complicated the firmware update procedure and limited the ability of the firmware to use the large data structures needed for receive packet filtering.

Shared RAM (shared by the CVAX processor and the LANCE chip) consists of another 256 kilobytes. This RAM contains the transmit and receive buffers as well as the LANCE transmit and receive rings. There is a vast amount of buffering space here, so the DEMNA device can tolerate a considerable amount of inattention from the host before being forced to discard incoming receive packets.

Erasable programmable read-only memory (EPROM) consists of 128K bytes for diagnostics and firmware boot code, including a backup copy of sufficient operational firmware to allow an update of EEPROM for initial load or subsequent update. EEPROM consists of 64K bytes for operational firmware, diagnostic patches, and error history data.

The gate array (data mover) handles the data move and quadword read/write operations. The data-move operations transfer buffers between the host and shared RAM. The quadword read/write

operations are used for control functions, such as reading ring entries, reading address translation information, and writing ring status on completion. Once the firmware initiates a data-move operation, other work is performed by the firmware while the data move progresses.

Interrupts are very costly; therefore, we chose to limit the number of interrupts fielded by the CVAX processor. A LANCE interrupt costs CVAX interrupt overhead, plus a LANCE CSR access, plus some normal interrupt overhead to save and restore registers. A data-move interrupt is less costly, but the firmware can be coded so that the data-move operation is usually complete, thus eliminating the need for the interrupt. Polling is performed for all LANCE- and data-move-related functions, but interrupts are used for local console I/O and error events.

Driver Design

The DEMNA team needed to design a driver that would be compatible with existing drivers but that would use all the features provided by the adapter. For VMS systems, this meant using the set of common routines that provide much of the data link functionality of the driver, but avoiding packet filtering. Another goal was to limit the copying of data by passing requests directly to the adapter.

For ULTRIX systems, the driver runs at a lower level with respect to packet filtering so it cannot take advantage of this feature. However, buffer chaining is used on the transmit side. As a transmit request traverses the various software layers, it accumulates buffer segments which the driver has to concatenate into a transmit frame. To avoid buffer copies in all but the extreme and infrequent cases, the driver then passes up to 11 buffer segments to the adapter.

To allow customer-written drivers for special applications, we documented the interface to make it readily available to customers.

Debug Tools

The adapter has a very simple mission in life: to transmit and receive packets. To verify operation, some debug tools are needed. The goal for the DEMNA team was to provide extensive debug tools both in the operational firmware and in standalone user tools. This design would allow debugging and verification in the development lab and in other, less-controlled environments. These debug tools are discussed further in the Visibility section.

Implementation

This section describes the implementation of the DEMNA adapter through its major functional blocks:

- Scheduler
- Port processing
- Command processing
- Transmit task
- Receive task
- Console task
- Monitor task

Scheduler

The scheduler is a round-robin routine that simply checks for work, does it, checks for work, does it, etc. There are no context switches, but some context is maintained in registers and shared by all routines. The scheduler, when idle, consists of about 18 VAX MACRO instructions. Transmit and receive tasks are given higher priority by duplicating their scheduler entry. When not idle, one pass of the scheduler processes four packets.

Port Processing

Port processing controls adapter initialization and shutdown, LANCE initialization and restart, fatal adapter error handling, gate array error handling, and miscellaneous host interface functions. This task also handles firmware updates of EEPROM.

Command Processing

The command ring usually contains transmit buffers, which can contain commands for special functions. These commands are included in the command ring to allow the port driver to synchronize control requests with transmit requests, e.g., user startup and stopping.

Command processing routines are called by the transmit task after the command buffer has been read from host memory. The commands consist of user startup (consisting of user context such as protocol type, packet format, physical address to use, and multicast addresses to enable), user stopping, read counters, and a set of maintenance commands.

Transmit Task

The transmit task copies a packet from the host memory to adapter buffer memory and tells the

LANCE to transmit it onto the Ethernet (store and forward). After the LANCE has completed the request, the firmware writes transmit status to the command ring entry, signifying completion of the transmit.

To minimize service time, the code in the transmit path was carefully scrutinized. The number of checks and branches was minimized for the optimized path. The optimized path through the transmit code is the 30-bit virtual addressing path, which is the most used. However, the 40-bit physical addressing path still results in better throughput because this path does not require any address translations, which are timely. The instruction sizes were shortened when possible, using word instructions instead of longword instructions, to reduce the amount of instruction prefetch by the CVAX processor. Routines were placed on quadword boundaries to maximize cache efficiency. When waiting for data moves to complete (getting the transmit buffer from host memory) or obtaining address translation information from the host, the firmware was designed to perform other functions to increase the probability that the operation would be performed when the firmware needed it.

Receive Task

The receive task has the simple job of handing received packets to the port driver. This task is complicated by the need to off-load the host of part of receive processing (including packet filtering, packet validation, maintenance of counters, and processing MOP messages) and to make duplicates of packets when more than one user has requested a copy. It is further complicated by the need to provide buffering, which the port driver uses to prevent the driver from supplying large numbers of buffers. For enhanced performance, the firmware deals with receive packets in small groups (192 bytes) to allow the benefit of cut-through on larger packets.

Packet filtering is done for the destination address and for user type, either protocol type for Ethernet, destination service access point (DSAP) field for 802, and protocol identifier (PID) value for 802 subnetwork access protocol (SNAP) packets. Additional filtering is done for users who request all traffic or all multicast traffic. Filtering is done by maintaining a 64-bit user mask, which accumulates the list of users who want a copy of the packet according to the characteristics of the packet and what each user has requested.

Packet validation consists of length checks for Ethernet frames (if the user is using a length field after the protocol type) and for 802 frames. This saves the driver a little work. Additionally, users can request only packets smaller than a selected size; the adapter discards packets that exceed this size.

The cut-through feature adds complexity and reduces throughput on small packets, but provides many benefits for larger packets. When a packet larger than 192 bytes is received, the packet filtering and validation of all but the length is done for the first segment. This segment is then copied into the host buffer, and subsequent segments are copied appropriately. The last segment completes the packet validation and cyclic redundancy check (CRC). The difficulty occurs when the packet validation fails or an error is detected, because the packet is discarded and the context for the now-free receive buffer has to be restored. The firmware elects to save as little context as possible for each packet and to regenerate buffer context after the error, i.e., fetching the ring descriptor anew and redoing the address translation.

Console Task

The console task accepts and parses console commands and displays the requested data. There are two means of accessing the console: local and remote. The local console is accessed by a terminal connected directly to the DEMNA adapter. The remote console is accessed through MOP console carrier commands directed at the adapter from another system. A remote console may also be used to access a DEMNA device on the local system (coming in through transmit instead of receive). The firmware does not distinguish between transmit or receive operations from remote consoles. The console block accepts the commands and decodes them, and the monitor block determines the status. The monitor block passes this status back to the console block where it is formatted and displayed on the screen.

Due to code size limitations in the EEPROM, compressed versions of the console screens are stored in the EEPROM. At initialization time the screens are uncompressed and stored in the RAM. (The screen compression saved 5 kilobytes in the EEPROM.) To easily setup and maintain the screens, especially since they often changed during the project, the screens were set up in separate text files. The fields in the screen were coded with different data types, such as date or longword. The screen was then put

through a PASCAL program to convert it to a VAX MACRO data structure and compress it.

The local console and the remote console can be run simultaneously. They have separate input and output buffers, the same decode and formatting code, and different input and output methods.

The remote console uses the MOP console carrier, coming in on transmit or receive. The command/poll and response/acknowledge commands are sent by the MOP program, i.e., either the network control program (NCP) or a user program that implements the MOP console carrier. The console code extracts the input characters from the command/poll packet and returns a response/acknowledge packet with any available data from the remote console output buffer. When a command has been entirely received, it is decoded and executed and the response placed in the remote console output buffer, which is sent back to the user in response/acknowledge packets.

The local console is a terminal directly connected to the DEMNA device and interfaced through the SSC universal asynchronous receiver transmitter (UART). This terminal connection receives and transmits one character at a time. Characters are collected into the local console input buffer and complete commands are parsed and executed. Response data is placed in the local console output buffer. The local console uses interrupts to signal when a character has been typed or when the UART is ready to transmit another character. These are the only interrupts used on the module, except for error interrupts. Since console interrupts are relatively infrequent, they are less costly than polling.

Monitor Task

The monitor facility operates mainly during receive or transmit. It also runs as a low priority entry in the scheduler to deal with debugging and verification activities (when debugging firmware is enabled).

Performance

As stated previously, the primary goal of the DEMNA adapter was high-speed performance, i.e., this adapter would not create a bottleneck when placed in a system. The major performance metrics we identified were throughput, service time, latency, and reliability.

- Throughput is the number of packets or bytes of packet data that can be transmitted or received per unit of time.
- Service time is the time a packet spends in each stage along its path from source through host software and driver, through adapter, over wire, through adapter, and through driver and host software to the destination.
- Latency is another measure of service time. It is a measure of delays encountered by queue depths of more than one at various points.
- Reliability is measured as the probability of packet loss under a receive load. It is also measured as adapter buffering and host buffer allocation effectiveness. For some protocols, recovery from packet loss takes a significant amount of time, and the loss of a packet may be quite noticeable to a user. Hence, recovery is related to a user's perception of reliable operation.

The performance goal of the DEMNA team was to minimize the service time through the adapter to maximize throughput. This is most critical for small packet sizes. If the service time is greater than the time it takes to transmit or receive a packet, then queue depths increase, increasing latency for subsequent packets. Small packets are critical because, obviously, they take less time to transmit or receive.

The speed of the Ethernet wire and the XMI bus must also be considered. The Ethernet operates at 10 megabits per second. The available bandwidth into memory and the capacity of the XMI are much greater; thus, the Ethernet is the limiting factor. To maintain maximum throughput, the DEMNA device must write and read packets to and from host memory at a speed equal to or greater than the Ethernet wire. If this speed is obtained, then the service time of the DEMNA adapter must be less than the time it takes to transmit or receive one 64-byte (small) packet to or from the Ethernet wire to maintain maximum throughput at all packet sizes.

Hardware

The primary hardware factors influencing adapter performance are CVAX performance, DMA engine throughput, and bus contention.

The gate array DMA engine can sustain between 11.5 and 13.5 megabytes per second on a VAX 6000 system. When transferring packet data (and attendant host ring processing), the firmware can sustain about 5.8 megabytes per second. This is the approximate rate at which the firmware would deliver a burst of large packets that had been stalled due to a lack of receive buffers.

The CVAX chip used is the 60-microsecond variant (the same one used in the VAX 6000 Model 310 processor). As seen in Figure 1, the processor runs on its own internal CDAL bus which has RAM containing firmware and private data structures. Thus the processor does not contend for the same bus as the gate array and the LANCE chip. However, the CVAX processor does touch shared memory and gate array registers; therefore the possibility of contention is significant. Logic analyzer measurements indicate that about 14 percent of CVAX cycles are consumed while waiting for access to the shared memory bus for minimum size packets. For large packets the consumption is 33 percent, but the cycles needed are considerably less than the remainder. The effect on the gate array accounts for part of the difference between the speeds of 11.5 to 13.5 megabytes per second and of the 5.8 megabytes per second mentioned above.

Firmware

Throughput is limited by the Ethernet bandwidth for packet sizes greater than 88 bytes. The average packet size on Ethernet is approximately 150 to 450 bytes per packet for a mix of DECnet, LAT, and cluster traffic. Table 1 represents the throughput that the host software can see, given sufficient host computes. These numbers show what might be expected. Virtual addressing costs some performance, and receive filtering accounts for most of the difference between transmit and receive.

It is interesting to look at the number of instructions executed by the CVAX processor for each receive and transmit packet as the measure of how much work must be done for each packet. These instruction counts are for minimum size packets in virtual address mode and increase slightly with increasing packet sizes.

For a transmit, the number of instructions required was about 134, consisting of 5 instructions for work done in the scheduler to determine initial transmit context, 77 instructions for the data transfer from host memory, 18 instructions to get the LANCE chip to begin transmitting, and 34 instructions to process packet completion and to update status in the transmit ring entry in host memory.

For a receive, the number of instructions required was about 160, consisting of 5 instructions for work done in the scheduler to determine initial receive context, 40 instructions to deal with the LANCE operations, 20 instructions for packet filtering, 65 instructions for the data transfer to host memory (including some time spent finding a user and validating the packet length), and 30 instructions for the prefetch of the next receive ring entry.

Some throughput was traded off in the interest of reducing adapter-added latency. By processing receive packets in groups of 192 bytes, the latency contribution for any packet size is much smaller than it would be if all the packet processing occurs after the packet has been fully received. Thus the time between the end of a packet on the wire and the host interrupt is fairly constant from 64- to 1518-byte packets, 50 to 70 microseconds.

Reliability

Reliability, or probability of loss, is measured by how large a burst of traffic the adapter can withstand at

Table 1 DEMNA Throughput

Packet Length (bytes)	Microseconds					
	Ethernet Maximum	LANCE Maximum	Transmit Virtual	Transmit Physical	Receive Virtual	Receive Physical
64	14880	14662	13181	14633	12468	12918
72	13586	13404	12592	13361	12254	12830
80	12500	12345	12247	12340	11813	12227
88	11574	11441	11432	11438	11441	11441
96	10775	10660	10656	10658	10660	10660
112	9469	9380	9380	9380	9380	9380
128	8445	8374	8374	8374	8374	8374
256	4528	4508	4508	4508	4508	4508
512	2349	2344	2342	2344	2344	2344
1024	1197	1195	1195	1195	1195	1195
1518	812	812	812	812	812	812

the maximum receive rate and deliver these packets to the host without losing any. Adapter reliability was measured at various packet sizes. A burst of 5 seconds without packet loss was considered to be of "infinite" duration.

Table 2 shows that the DEMNA adapter can survive a significant burst of activity without packet loss. Such activity is unlikely, but possible, depending on the application being run and on the network configuration.

This testing does not measure how host software performs buffer allocation for a user application or for the adapter as a whole. For the latter, the DEMNA adapter accounts for any lack of buffering by the host by not discarding a packet if a buffer is not immediately available. Instead, it waits up to three seconds for the host to supply a buffer.

Visibility

A system user looking at the operation of the network sees three areas of complexity: the system software, the network controller, and the network. When everything is working well, there is little need to look at any of these areas except perhaps to predict future operation (by extrapolating network utilization or system usage) or to confirm that the system is indeed running well. When the system is not running well, visibility into these areas is crucial to understanding what is wrong and how to correct it. The console and monitor facilities were built into adapter firmware from the outset; we knew that the visibility was crucial to adapter debugging and verification and would later be helpful to users.

System Operation

The console displays XMI utilization as apportioned among the XMI devices. This data comes from sampling done by the firmware of the "last XMI node active on the bus." From this, the user can estimate total XMI utilization.

The console also displays buffer occupancy on the adapter for transmit and receive, user configuration as to protocol type and characteristics, buffer availability counters, and host interrupt counters. This data indicates how the system is running, i.e., whether sufficient buffers are allocated to the device and to each user of the device. These counters also indicate how much attention the driver is paying to the adapter. For example, if the system is not tuned properly, the adapter may be generating less than normal interrupts (because queuing delays are affecting the system operation). These queuing delays can be seen in the firmware counters, which monitor the depth of adapter queues and the ability of the adapter to give receives to the host, i.e., buffering on the adapter has been used to compensate for queuing delays in the host.

Adapter Operation

When the adapter is not malfunctioning, visibility into adapter utilization is important. The console displays program counter (PC) sampling results for the firmware, showing how busy the adapter is and where time is being spent. When looking at the I/O subsystem as a whole, it is important to know how much the adapter is contributing to queuing delays, buffer occupancy, and added latency. This adapter operation can be seen by looking at how busy the adapter is and how many buffers it has outstanding.

For adapter failure or problems on the XMI, the console displays error information which has been saved in EEPROM. This error data consists of fatal error context, data transfer or XMI error context, and results of self-test.

Network Operation

The DEMNA device normally sees all packets on the wire (excluding packets less than 64 bytes in length [runt packets] and collision fragments). When looking at the adapter operation through the console facility, the user sees current network utilization

Table 2 DEMNA Receive Burst Tolerance

Packet Length (bytes)	Burst Virtual (packets)	Burst Virtual (microseconds)	Burst Physical (packets)	Burst Physical (microseconds)
64	3250	221661	3843	262106
72	5116	381677	11591	864741
80	9917	803321	Infinite	Infinite
88	Infinite	Infinite	Infinite	Infinite

and network error information. For transmit errors, the console displays the number of errors and date and time of the last occurrence. For receive errors, the console displays the number of errors, date and time, source address, and protocol type. Additionally, receive errors that are not counted (because they do not pass receive filtering) are displayed. For example, error information is displayed for a node generating packets with CRC errors regardless of the destination of these packets.

The console also provides the command SHOW NETWORK to display network utilization in node addresses and protocol types. For this command, the receive firmware calls a monitor facility routine for each packet seen on the wire. This routine maintains statistics for each source and destination node address, consisting of the number of packets and the number of bytes. At three-second intervals, the console calls a monitor routine which adds statistics over the prior interval to cumulative data for each node, collects top nodes and protocol data, and clears the interval data to prepare for the next three seconds of monitoring. Figure 3 represents a sample network monitoring display.

Debug Tools

The monitor task provided other debugging functions during adapter debugging and internal field test. These functions are not visible features in the finished product. However, they are extensions to the functionality and illustrate the benefits of visibility into the adapter. A user program, XNAMON, was written to access the following functions.

- Traffic generation. It is difficult to generate heavy loads on an adapter, particularly because of logistics. Other systems are needed with enough processing power to generate the load. Using the XNAMON program, only one system was needed. XNAMON was run on it to direct other adapters to generate traffic to another node with a particular packet size at a specified rate. Since traffic generation could be done

```
 — 19.117          — Network —  21-APR-1991 11:29:38 —

                — 3000002 usecs —   21.6% NI —   — 00:00:33 —   19.7% NI —
#   User          Pks/Sec   Byt/Pk   %NI-Cur    Packets    Bytes(k)  %NI-Tot
-   ------------  -------   ------   -------    -------    --------  -------
1   60-07 NISca       571      214     10.7%      15019        3041     8.1%
2   60-03 DECnet      177      645      9.4%       6358        4021    10.0%
3   60-04 Lat         167       64      1.1%       5765         379     1.2%
4   60-02 MopRC        18       87      0.1%        659          56     0.1%
5   80-41 LAST          7       82      0.0%        206          17     0.0%
6   FE-00               2      271      0.0%         54          20     0.0%

#   Nodes         Pks/Sec   Byt/Pk   %NI-Cur    Packets    Bytes(k)  %NI-Tot
-   ------------  -------   ------   -------    -------    --------  -------
1   19.187            177      631      9.2%       5177        3530     8.8%
2   63.631            251      244      5.3%       6402        1551     4.0%
3   19.52             170      356      5.1%       3321        1410     3.5%
4   19.209             55      615      2.8%       2272        1455     3.6%
5   19.167             60      558      2.8%       2012         982     2.4%
6   19.117            112      300      2.8%       3227         873     2.2%
7   63.619            119      252      2.6%       2076         514     1.3%
```

KEY:

usecs	MICROSECONDS
NI	NETWORK INTERCONNECT (TRANSMISSION MEDIUM)
%NI	PERCENTAGE OF AVAILABLE BANDWIDTH UTILIZED
NISca	NETWORK INTERCONNECT SYSTEM COMMUNICATIONS ARCHITECTURE
MopRC	MAINTENANCE OPERATIONS PROTOCOL REMOTE CONSOLE
LAST	LOCAL AREA STORAGE TRANSPORT
Pks/sec	PACKETS PER SECOND
Byt/Pk	BYTES PER PACKET

Figure 3 Network Monitoring Display

regardless of system state (except for power on), there was always a good supply of traffic generators.

- Packet tracing. This function allowed a node to scan the receive stream for packets with selected source and destination addresses and protocol types. Either the packet header or the entire packet was saved for matching packets. This function was used extensively during initial debugging for validating transmit functionality. Later it was used for validation of MOP and related functionality by creating trace files on a known good node. We then ran functional scripts through a test generator, which used the traffic generator on one node to send a test packet to the node under test. The command and the response were traced by the trace node and the test program collected the trace data and compared it against known good data. Packet tracing was also used to verify packet filtering by devising a test program that could start up particular user configurations and loop back any packets received.

- Adapter test. The ability to exercise a module under stress was critical to adapter hardware verification. The functionality in question was the Ethernet subsystem and the XMI interface through the gate array. The monitor facility provided this test functionality by doing MOP loopback operations to another node while doing various data transfer operations to host memory. Data compares were done on completed transactions to validate data integrity. The XNAMON program provided the interface for this function and the remote display of its results.

- Remote debugger. The access to DEMNA internals allowed remote adapter memory dumps and remote inspection of data structures while the adapter was running.

Conclusion

The DEMNA adapter meets the requirements of the VAX 6000 and VAX 9000 systems. In fact, the performance for small packets exceeds the capability of these systems. For larger packets, Ethernet bandwidth is the limiting factor. Our experience illustrates some advantages and disadvantages of choosing a firmware-based design over an interface implemented entirely in hardware.

Advantages of a Firmware-based Design

The advantages of designing an adapter in firmware are as follows:

- The firmware can usually off-load host computes by doing more pre-processing.

- The firmware can be changed easily (bug fixes or changes in functionality), thus reducing long-term maintenance and support costs. Also, changes can be made in the field by a firmware upgrade rather than requiring module rework at a manufacturing site.

- By designing in the firmware, designers can avoid software driver complexity and the necessity of hardware redesign.

- The firmware can provide powerful debugging mechanisms and tools.

- The firmware is very flexible. Changes to support hardware problems or additional off-load of host computes can be considered late in the design cycle. This may also allow new port architecture and addressing changes for creating new products.

- Firmware designs allow extensive functionality for lower product and development cost than a total hardware design.

- Firmware designs allow the hardware to be released earlier in the development cycle.

Disadvantages of a Firmware-based Design

The disadvantages of designing an adapter in firmware are:

- The adapter is generally more expensive, considering the cost of a microprocessor subsystem with enough computes for the job.

- The adapter is slower in terms of latency. Some applications may be more sensitive than others, given the same throughput, but may have slightly larger service times per packet. The effect can be viewed in terms of buffer occupancy: an adapter with lower latency may utilize, on average, few buffers.

- The approach is not feasible for transmission media much faster than Ethernet, because the performance requirements of the microprocessor

become extreme or the hardware assists for the microprocessor become too complex and costly.

Future Directions

Several characteristics distinguish future anticipated system design from current systems (such as the VAX 6000 and VAX 9000 systems).

- Increased host processor power
- Simplified bus design
- Increased I/O bandwidth requirements

Increased host processor speed moves the I/O bottleneck from the host to the I/O subsystem. To supply the I/O needs, the I/O subsystem must provide faster media, e.g., fiber distributed data interface (FDDI) in the near term, or multiple connections to slower media (such as Ethernet). The I/O adapters will be expected to provide significantly greater throughput with a smaller adapter contribution to latency. The effective performance of the system will be more sensitive to latency. For example, an application using a single threaded command/response protocol is extremely dependent on the amount of service time through the I/O subsystem at each end. As the processing speed increases, application overhead is reduced and throughput becomes dominated by the service time of the adapter and the transmission time.

Faster processors place a greater burden on the system bus and I/O interface, which necessitates a simpler bus protocol. This might consist of eliminating costly functionality such as byte masking and interlocks. However, a simpler interface to the I/O adapter will require considerable change to the port protocol to ensure its efficiency.

The characteristics needed in future adapters are as follows:

- Greater throughput. This means more connections to a slower medium, such as a single adapter supporting multiple Ethernet connections. Or it means a faster medium. Additionally, configurations using Ethernets as point-to-point links will be more common, thus implying a heavier load on each Ethernet.
- Simpler host interface. This is necessitated by the simpler bus protocol. Bus overhead should be minimized, which includes the elimination of such functionality as page table access for virtual address translation. Also, the bus transfer size used by the adapter should be compatible with the basic data size of the system to avoid cache thrashing and unnecessary read-modify-write transactions.
- Reduced latency. The adapter should minimize its contribution to transmit and receive latency. This may mean reducing some of the functions done by an intelligent adapter on receive, in order to speed delivery to the host after packet reception is complete. These functions include packet filtering, handling of maintenance operations packets, length validation, and maintaining counters data. Improving packet filtering by host software would eliminate the reason for placing this function on the adapter in the first place.

  Filtering in host software is considerably more difficult than in the adapter. The difficulty comes from the need to deal with extreme user configurations. The DEMNA is bounded by limiting the users and node addresses. The extreme cases must still be done by host software.

Acknowledgments

The authors would like to acknowledge the following members of the DEMNA design team: Barbara Aichinger, Keith Bilafer, Mark Cacciapouti, Don Dossa, Linda Duffell, Bernie Hall, Jeff Huber, Helen McGreal, Jonathan Mooty, Dave O'Keefe, David Oliver, Brian Parr, Art Singer, Andy Stewart, Fred Templin, Vicky Triolo, Ed Tulloch, and Don Villani.

General References

DEC LANcontroller 400 Programmer's Guide (Maynard: Digital Equipment Corporation, Order No. EK-DEMNA-PG-001, 1990).

DEC LANcontroller 400 Console User's Guide (Maynard: Digital Equipment Corporation, Order No. EK-DEMNA-UG-001, 1990).

D. Mirchandani and P. Biswas, "Ethernet Performance of Remote DECwindows Applications," *Digital Technical Journal*, vol. 2, no. 3 (Summer 1990): 84–94.

D. Boggs et al., "Measured Capacity of an Ethernet: Myths and Reality," *Proceedings of SIGCOMM '88* (ACM SIGCOMM, 1988): 222–234.

The Ethernet: A Local Area Network, Data Link Layer and Physical Layer Specifications, Version 2.0 (Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, Order No. AA-K759B-TK, 1982).

Satish L. Rege

The Architecture and Implementation of a High-performance FDDI Adapter

With the advent of fiber distributed data interface (FDDI) technology, Digital saw the need to define an architecture for a high-performance adapter that could transmit data 30 times faster than previously built Ethernet adapters. We specified a first generation FDDI data link layer adapter architecture that is capable of meeting the maximum FDDI packet-carrying capacity. The DEC FDDIcontroller 400 is an implementation of this architecture. This adapter acts as an interface between XMI-based CPUs, such as the VAX 6000 and VAX 9000 series of computers, and an FDDI local area network.

Fiber distributed data interface (FDDI) is the second generation local area network (LAN) technology. FDDI is defined by the American National Standards Institute (ANSI) FDDI standard and will coexist with Ethernet, the first generation LAN technology.

The architecture and implementation presented in this paper are for the DEC FDDIcontroller 400, Digital's high-performance, XMI-to-FDDI adapter known as DEMFA. This adapter provides an interface between an FDDI LAN and Digital's XMI-based CPUs, presently the VAX 6000 and VAX 9000 series of computers.[1,2] DEMFA implements all functions at the physical layer and most functions at the data link layer.[3,4]

We begin the paper by differentiating between an architecture and an implementation. Then we present our project goal and analyze the problems encountered in meeting this goal. Next we give a historical perspective of Digital's LAN adapters. We follow this discussion by describing in detail the architecture and implementation of DEMFA. Finally, we close the paper by presenting some results of performance measurement at the adapter hardware level.

Adapter Architecture and Implementation

Before we discuss the DEMFA architecture and its implementation, it is necessary to understand what is meant by an adapter architecture and an implementation of that architecture. An adapter architecture specifies a set of functions and the method of executing these functions. An implementation that incorporates all of these functions and conforms to the method of executing these functions becomes a member of the adapter architecture family. Thus, for a given architecture, many implementations are possible.

To grasp the concept presented in the previous paragraph, consider the VAX CPU architecture. This architecture defines the instruction set, which is composed of a set of arithmetic, logical, and other functions, and a format for the instruction set that a processor should implement to be classified as a VAX computer. Examples of VAX implementations are the VAX 11/780 and the VAX 9000 computers, which both conform to the VAX CPU architecture.

Our Goal and the Problem Definition

Our goal was to define an architecture for an FDDI adapter that meets the ultimate performance goal of transmitting approximately 450,000 packets per second (packets/s). This goal is considered ultimate because 450,000 packets/s is the maximum packet-carrying capacity of FDDI. Note that this transmission rate is approximately 30 times greater than that of Ethernet, which can transmit approximately 15,000 packets/s.

Before defining the problem, the basic properties of XMI and FDDI must be understood. XMI is a

64-bit-wide parallel bus that can sustain a 100-megabyte-per-second (MB/s) bandwidth for multiple interfaces.[5] Each interface attached to the XMI bus is referred to as a commander when it requests data or a responder when it delivers data. XMI is an interconnect that can have transactions from several commanders and responders in progress simultaneously.

FDDI is a packet-oriented serial bus that operates using the token ring protocol and has a bandwidth of 100 megabits per second (Mb/s).[6] FDDI is capable of transmitting packets as small as 28 bytes, which take 2.24 microseconds to transmit. Therefore, FDDI can carry approximately 450,000 minimum-size packets/s. The largest packet that FDDI can carry is 4508 bytes. The ANSI/IEEE 802.5 standard defines the FDDI operation; Digital has developed its own implementation of the FDDI base technology as a superset of the ANSI standard.[3]

Our problem was to architect an adapter that could interface XMI, i.e., a parallel high-bandwidth CPU bus for VAX computers, to a serial fiber-optic networking bus. To avoid being the bottleneck in a system, such an adapter must be able to transmit or receive 450,000 packets/s.

ANSI defines the protocol for interfacing an adapter to an FDDI LAN.[6] But we had to define the protocol between the adapter and the VMS and ULTRIX operating systems used by most VAX computers. Thus, solving the problem required us to architect a data link layer adapter that would satisfy both protocols and meet the FDDI maximum packet transfer capability.

Historical Perspective

The computer industry has built many LAN adapters since the inception of Ethernet ten years ago. The first LAN adapter built by Digital was the UNIBUS-to-NI adapter (UNA). (NI is Digital's alias for Ethernet.) The Digital Ethernet-to-XMI network adapter, known as DEMNA, is Digital's most recent Ethernet adapter.[7]

Let us choose the maximum throughput rate expressed in packets per second as a performance metric for LAN adapters. The historical perspective shows that the first adapter to meet the Ethernet packet-carrying capacity is the DEMNA. Therefore, it took approximately eight years and six generations for an Ethernet adapter to achieve this throughput rate. Consequently, many designers thought that our goal of meeting the ultimate FDDI packet-carrying capacity was impossible.

But the DEMFA architecture, a first generation FDDI data link layer adapter architecture, can meet the maximum FDDI packet-carrying capacity. In this sense, the DEMFA architecture is ultimate.

Traditional Adapter Architectures

In this section, we analyze the traditional adapter architecture and show that by using this architecture we could not meet our performance goal. Figure 1 is a block diagram of a traditional adapter, e.g., DEMNA. In such a design, a CPU in the adapter operates on every transmitted and received packet. Thus, using this traditional architecture to build an ultimate FDDI adapter would require a CPU capable of handling 450,000 packets/s. To predict the performance of such a CPU, we extrapolated from the performance data of the CPU used in DEMNA.[7] This traditional adapter can handle approximately 15,000 packets/s using a CPU rated at 3 VAX units of performance (VUPs).

If we assume a linear model to extrapolate the performance of a CPU from DEMNA to DEMFA, an

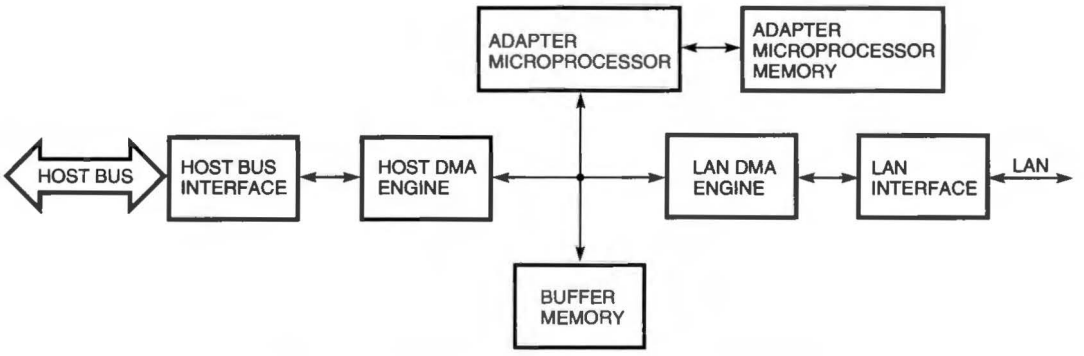


Figure 1 Block Diagram of a Traditional Adapter

ultimate FDDI adapter would require at least a 90-VUP CPU. Such a CPU was neither available nor cost-effective for timely shipment of our adapter. Besides, it would be extravagant to use a 90-VUP CPU in an adapter whose host CPU may have a performance as low as 3 to 4 VUPs. Therefore, we looked for a different solution.

DEMFA Architecture

The DEMFA architecture is characterized by the following specifications for functionality and the means to achieve this functionality:

- As mentioned earlier, the DEMFA architecture implements all functions at the physical layer and a major subset of the functions at the data link layer.
- The architecture requires that this functionality be implemented in pipelined stages, which are used to receive and transmit packets over the FDDI ring without CPU interference.
- The DEMFA architecture specifies a ring interface for communicating between the pipelined stages. Rings operate as queues that allow buffering between pipelined stages, enabling these stages to proceed in an asynchronous fashion.
- The architecture requires a packet-filtering capability in the pipelined stage nearest to the FDDI ring; this capability helps to minimize adapter and host resource utilization.
- The architecture specifies the DEMFA port, which minimizes the information transfer required to interact with the host operating system. This interaction takes place during both initialization and the normal operation of receiving and transmitting packets.

In the following sections, we elaborate on different features of the DEMFA architecture.

Pipelined Architecture with No CPU Interference

Once we determined that the traditional architecture of a CPU processing the packets could not meet our performance goal, we began to investigate alternative architectures. The requirement was to either process one receive packet or queue one transmit packet in a time period less than or equal to the time it takes to transmit on an FDDI ring. Thus, the device we architected must process 28-byte packets in less than 2.24 microseconds. A little thought will show that if we are able to meet the requirements for processing small packets at the FDDI bandwidth, then the requirements for larger packets can be easily met.

Our final choice was a three-stage pipeline approach which broke down the complexity of implementation while meeting our performance goal. As shown in Figure 2, the three stages of the pipeline in the adapter are the FDDI corner and parser (FCP) stage, the ring entry mover (REM) stage, and the host protocol decoder (HPD) stage. Figure 2 also shows two other functions required of the adapter: the buffering of packets, which requires a memory called the packet buffer memory (PBM) and a memory interface called the packet memory interface (PMI); and the local intelligence, also called the adapter manager (AM).

DEMFA Functions

This section presents brief descriptions of the DEMFA functions and the pipelined stages in which these functions are performed. This, according to our definition, is the DEMFA architecture. A later section, One Implementation of the DEMFA Architecture, describes an implementation in detail.

The FCP stage converts serial photons on the FDDI ring into packets and then writes the packets into PBM longwords, 32 bits at a time. The parser implements the logical link control (LLC) filtering functionality. This stage is also responsible for capturing the token on the FDDI ring, transmitting packets, and implementing the physical layer, e.g., media access control (MAC), functionality required by the FDDI standard.

The REM stage is responsible for distributing packets received over the FDDI ring to the host computer and to the AM. This stage also collects the packets from the host and the AM to queue for FDDI transmission.

The HPD stage interfaces with the XMI bus to move received packets from PBM to the host memory and to move transmit packets from the host memory to the PBM.

The PBM stores the packets received over the FDDI ring and the packets to be transmitted over the FDDI ring. It also stores the control structures required for accessing these packets. The PMI arbitrates the requests made by the three pipelined stages and the AM to access the PBM.

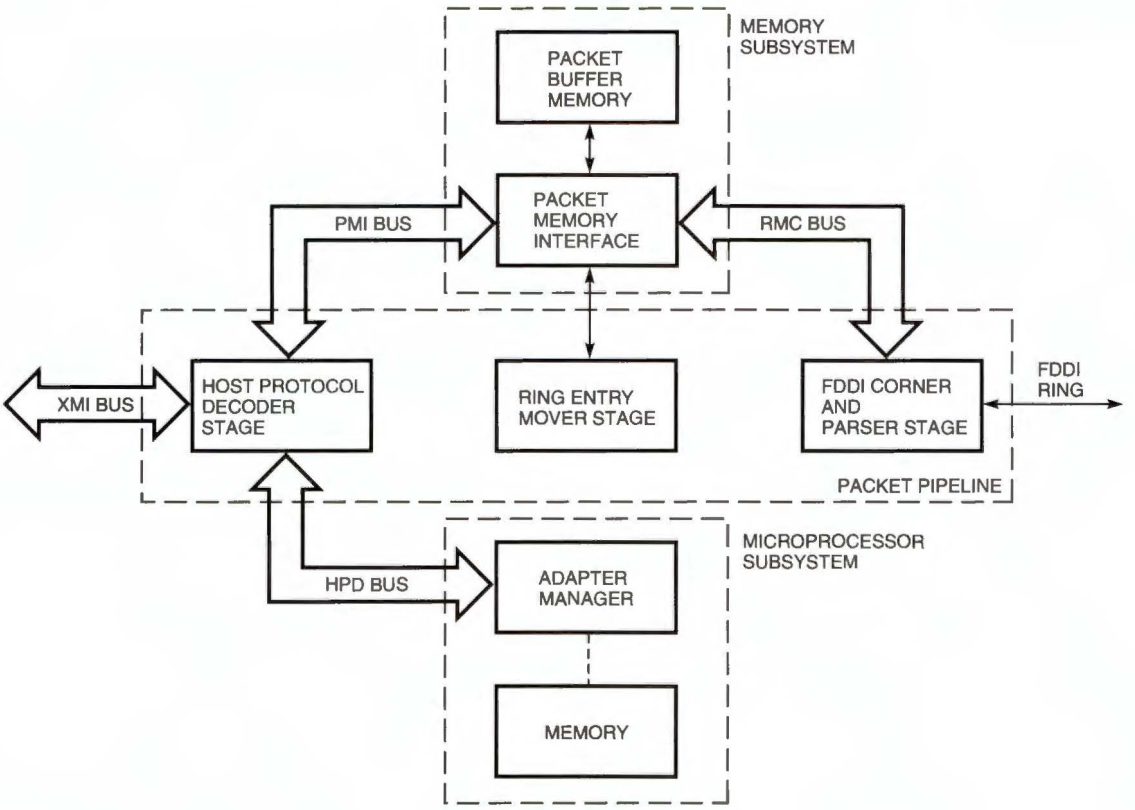


Figure 2 DEMFA Block Diagram

The AM implements the functionalities of self-test and initialization in the adapter and also a subset of the SMT function required by the ANSI FDDI specification.[8] The adapter manager performs no function in either the receipt or transmission of individual packets to the host.

We use ring interfaces to communicate within the adapter and between the adapter and the host. These interfaces are described in detail immediately following the next section.

Performance Constraints on the Pipelined Stages

Consider the three pipelined stages and their ring interfaces. At any time, the three independent stages are processing different packets. Thus, if the HPD stage is processing received packet 0, the REM stage may be working on received packet 4 and the FCP on received packet 7. Note that packets 1, 2, and 3 wait on a ring between the REM stage and the HPD stage. Similarly packets 5 and 6 wait on a ring between the FCP stage and the REM stage. The PBM must have enough bandwidth to service the three stages. It also must service them with low latency so that the first-in, first-out (FIFO) buffers in the FCP stage do not overflow.

By dividing the processing of a packet over the three stages and the ring interfaces used to queue packets between these stages, we reduced the complexity of the total adapter functionality. Any implementation of this architecture specification would consist of three loosely coupled designs that use ring interfaces to communicate with one another.

Each stage must process a packet in less time than it takes to transmit the packet on the FDDI ring. As we mentioned previously, this transmission time is 2.24 microseconds for the smallest packet. A larger packet may take longer to process than a small packet, but such a packet also takes longer to transmit on the FDDI ring.

Thus, to meet our performance goal, we architected a three-stage pipeline implementation, with each stage meeting a packet-processing time

dependent upon the packet size. In addition, our architecture specified a PBM with sufficient memory bandwidth to service the asynchronous requests from the three stages with minimal latency.

Ring Interface—The Core of the DEMFA Architecture

The ring interface forms the core of the DEMFA architecture. An interface is necessary to exchange data between the adapter and the host computer and also between the different stages and functional units of the adapter. Such an interface usually consists of a data structure and a protocol for communication. We evaluated various data structures, including a linked list or queue data structure, and found that a ring data structure is efficient to manipulate and would be easy to implement in state machines, if desirable.

Implementation of Ring Structures Ring structure implementation requires a set of consecutive memory addresses, as shown in Figure 3. The ring begin pointer and the ring end pointer define the beginning and end of a ring. Two entities, the transmitter and the receiver, interface with a ring to exchange data. The transmitter interface delivers data to the receiver interface using the ring structure. This data resides in memory that is managed by one of the two interfaces. If the transmitter interface manages the memory, the ring is called a transmit ring. If the receiver interface manages the memory, the ring is called a receive ring.

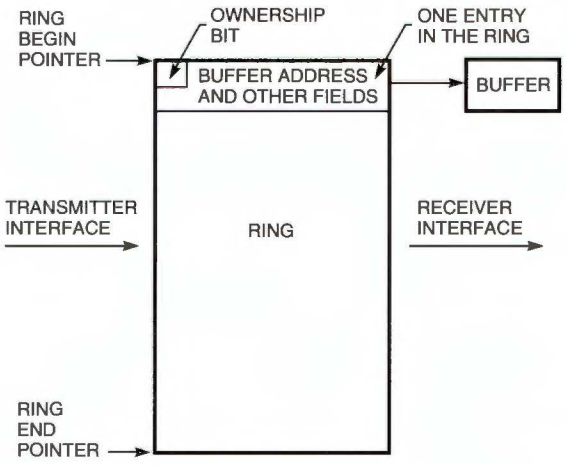


Figure 3 Ring and Ring Interfaces

Rings are divided into entries that consist of several bytes each; the number of bytes in an entry is an integral multiple of longwords. A ring, in turn, must contain an integral number of entries. The entry size and the number of entries in a ring determine the ring size. We chose an entry size that is a power of two in bytes and the number of ring entries to be divisible by two, as well. These choices helped to simplify the hardware implementation used to peruse these rings.

Each entry consists of

- An ownership bit, which indicates whether the transmitter interface or the receiver interface owns the entry
- Buffer pointers, which point to transmitted or received data
- A buffer descriptor, which contains the length of the buffers, and status and error fields

The definitions of these fields in an entry and the rules for using the information in these fields constitute the ring protocol.

Only the interface that owns an entry has the right to use all the information in that entry. This right includes using the buffer pointers to operate on data in the buffers. Both interfaces have the right to read the ownership bit, but only the interface with ownership may write this bit.

The two interfaces can exchange entries by toggling the ownership bit. After toggling this bit, the transmitter and receiver interfaces need to prod each other to indicate that the ownership bit has been toggled. This is accomplished using two hardwired Boolean values, by means of an interrupt, or by writing a single-bit register. Hardwired Boolean values are used when both the transmitter and the receiver are on the adapter. Either the interrupt scheme or the method of writing a single-bit register is used when the transmitter and receiver converse over an external bus, e.g., an XMI bus.

The word "signal" is used henceforth to represent the prodding of one interface by the other. A transmitter interface uses "transmit done" to signal the receiver interface that data has been transmitted. A receiver interface uses "receive done" to signal the transmitter interface that the data has been received. Note that we have defined the DEMFA port protocol in such a way that the number of interrupts used to signal the host across XMI is minimized to reduce the host performance degradation caused by interrupts.

The unit of data exchanged between the transmitter interface and the receiver interface is a packet. A packet may be written in a single buffer if the packet is small or over multiple buffers if the packet is large. In this paper, we use the term buffer to refer generically to buffers in the adapter or in the host. The buffers in the adapter are always 512 bytes in size and, when referred to specifically, are called pages. The buffers in the host may be of different sizes.

An exchange of data requires single or multiple buffers, depending upon the packet and buffer sizes. One field of two bits in the buffer descriptor is used to designate the beginning and end of packet. These bits are called the start of a packet (SOP) and the end of a packet (EOP). Thus, for a one-buffer packet both the SOP and the EOP are asserted. For a multiple-buffer packet, the first buffer has the SOP asserted, the middle buffers have both the SOP and the EOP deasserted, and the last buffer has the EOP asserted. The buffer descriptor also contains fields that we do not describe in this paper.

Data Exchange on a Transmit Ring Data exchange between a transmitter interface and a receiver interface is accomplished in a similar manner on both transmit and receive rings. Therefore, we discuss the exchange in detail for a transmit ring; for a receive ring, we note only the dissimilarities.

The events that occur during the data exchange on a transmit ring are shown in Figure 4. The process is as follows. The transmitter interface manages the memory used to exchange data and has two pointers to the ring entries, i.e., the fill pointer and the transmitter free pointer. The transmitter interface uses the fill pointer to deliver data to the receiver interface. The transmitter interface uses the transmitter free pointer to recover and manage the buffers freed by the receiver interface. The receiver interface uses only one pointer, i.e., the receive pointer, which points to the next entry that the receiver interface interrogates to receive data.

To understand how data is transmitted, assume that the pointers move from top to bottom, as shown in Figure 4. Initially, all the pointers designate the location indicated by the begin pointer.

A transmitter that has data to transmit to a receiver uses the entry indicated by the fill pointer. First, the transmitter verifies that it owns the entry by checking the ownership bit. Second, the transmitter writes the buffer address and the remaining fields in the entry. In the case of a single buffer packet, the transmitter interface writes a single entry and then toggles the ownership bit and signals the receiver interface.

For multiple buffers, the transmitter interface increments the fill pointer and repeats the two steps described in the previous paragraph to write all the buffer addresses and the length and status information. Then the transmitter interface toggles the ownership bits of all later entries of the multiple buffers before toggling the ownership bit of the first entry. This protocol preserves the atomicity of the packet transfer between the transmitter and

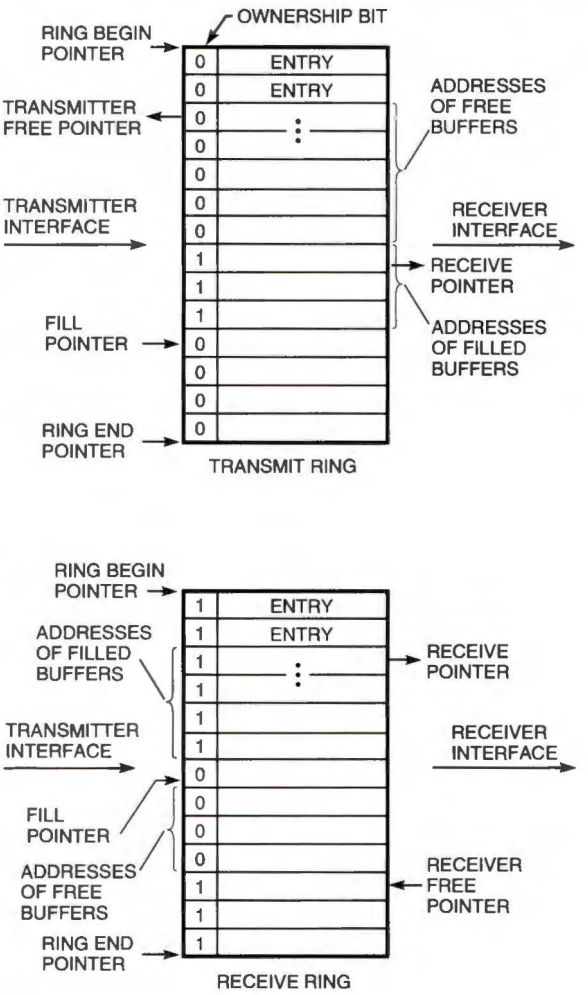


Figure 4 Data Exchange on Transmit and Receive Rings

receiver interfaces. Then the transmitter interface signals the receiver interface that a packet is available on the transmit ring. This signal alerts the receiver interface, which then examines the entry pointed to by the receive pointer. The receiver interface operates on the entry data if it owns the entry.

The receiver interface returns the entries to the transmitter interface by toggling the ownership bits and then signals receipt of data to indicate the return of the entries (and hence the free buffers). Note that there is no need to return these free buffers in a packet, atomic fashion. The transmitter interface uses the transmitter free pointer to examine the ownership bits in the entry and to reclaim the buffers.

The interfaces operate asynchronously, since each one can transmit or receive data at its own speed. If the transmitter interface can transmit faster than the receiver interface is able to receive, the transmit ring fills up. Under such circumstances, the receiver interface owns all the entries in a transmit ring, the fill pointer equals the transmitter free pointer, and data transmission stops. Conversely, if the receiver interface is faster than the transmitter interface, the transmit ring will be nearly empty. In this case, the transmitter free pointer and the receive pointer are almost always equal.

Note the following invariants that apply to the pointers when data is exchanged on a transmit ring: the fill pointer cannot pass the transmitter free pointer; the transmitter free pointer cannot pass the receive pointer; and the receive pointer cannot pass the fill pointer.

Data Exchange on a Receive Ring As also shown in Figure 4, the operation of data exchange on a receive ring is similar to that operation on the transmit ring, with the following differences. The receiver interface manages the memory used for exchanging data. Consequently, the receiver interface has two pointers, the receiver free pointer and the receive pointer, and the transmitter interface has only one pointer, the fill pointer.

Table 1 shows the various DEMFA rings and the transmitters and receivers that interface with each ring.

Table 1 DEMFA Rings and Their Transmitter and Receiver Interfaces

Rings	Transmitter	Receiver	Remarks
Rings in Packet Buffer Memory			
RMC Receive Ring	FDDI Corner and Parser Stage	Ring Entry Mover Stage	Contains data that originated on the FDDI ring.
RMC Transmit Ring	Ring Entry Mover Stage	FDDI Corner and Parser Stage	Contains data that originated at the host or the AM, destined for the FDDI ring.
HPD Receive Ring	Host Protocol Decoder Stage	Ring Entry Mover Stage	Contains data that originated at the host, destined for the FDDI ring.
HPD Transmit Ring	Ring Entry Mover Stage	Host Protocol Decoder Stage	Contains data that originated at the FDDI ring, destined for the host.
AM Receive Ring	Adapter Manager	Ring Entry Mover Stage	Contains data that originated at the AM, destined for the FDDI ring or the host.
AM Transmit Ring	Ring Entry Mover Stage	Adapter Manager	Contains data that originated at the FDDI ring, destined for the AM.
Rings in Host Memory			
Host Receive Ring	Host Protocol Decoder Stage	Host	Contains data that originated at the FDDI ring or the AM, destined for the host.
Host Transmit Ring	Host	Host Protocol Decoder Stage	Contains data that originated at the host, destined for the FDDI ring.
Command Ring (Transmit Ring)	Host	Adapter Manager	Contains commands that originated at the host for the AM; note that the AM replies in the same ring.
Unsolicited Ring (Receive Ring)	Adapter	Host Manager	Contains unsolicited messages from the AM to the host.

Subsystem Level Functionality

The basic functions that an FDDI LAN adapter is required to perform are receiving and transmitting packets over the FDDI ring. The adapter must be able to establish and maintain connection to the FDDI network. The connection management (CMT) protocol, a subset of the station management (SMT) protocol, specifies the rules for this connection.[8]

The implementation of the complex CMT algorithm in an adapter requires an intelligent component, such as a microprocessor, that can receive, interpret, and transmit packets. Note that the number of CMT packets that flow over the FDDI ring constitutes only a small fraction of the normal traffic. Therefore, a low-performance CPU is adequate to implement connection management. The CPU in the DEMFA device is called the adapter manager.

The packets in the receive stream that originated on the FDDI ring and are addressed to this host or adapter (together called the node) can take one of the following paths:

- Packets not addressed to this node are forwarded over the FDDI ring.
- Packets addressed to this node are delivered to the host computer.
- Packets addressed to this node are delivered to the AM.

The delivery of packets to the host computer implies that the adapter has a pointer to a free memory buffer in which to deposit the received packet. The DEMFA port, described in the next section, specifies the rules for extracting free buffer pointers from the host memory.

For each packet that the host needs to transmit, the adapter must know the buffer address or addresses and the extent of each buffer. The DEMFA port defines the method to exchange this buffer information. In addition, the host and the adapter microprocessor must be able to exchange information. The DEMFA port defines the protocol for this communication also.

DEMFA Port

The DEMFA port specifies the data structure and protocol used for communication between the adapter and the host computer. Rather than invent a new protocol, we modified the DEMNA port specification.[7] The data structure used to pass information between the host and the adapter is a ring structure. Such structures are more efficient to traverse than queue structures.

The DEMFA port defines the four separate host rings listed in Table 1:

- The host receive ring, which contains pointers to free buffers into which a packet received over the network can be deposited
- The host transmit ring, which contains pointers to filled buffers from which packets are removed and transmitted over the FDDI ring by the adapter
- The host command ring, which sends commands to the AM
- The unsolicited ring, which the AM uses to initiate communication with the host CPU

By using four host rings, we were able to differentiate between the fast and frequent data movement to and from the FDDI ring and the comparatively slow and infrequent data movement required for communication with the AM.

One Implementation of the DEMFA Architecture

Previous sections specified the DEMFA architecture. The remainder of this paper describes an implementation of the DEMFA architecture. In the following sections, we present details of the implementation for the packet buffer memory and the packet memory interface; the three pipelined stages, FCP, REM and HPD; and the adapter manager.

Packet Buffer Memory and Packet Memory Interface

The packet buffer memory stores the data received over the FDDI ring before delivering this data to the host. The PBM also stores data from the host before transmitting over the FDDI ring.

PBM consists of two memories: the packet buffer data memory and the packet buffer ring memory. Virtually, the packet buffer data memory divides into seven areas—one used by the AM and three each for data reception and data transmission to and from the three external interfaces. These three interfaces are the FCP stage, the HPD stage, and the AM. The areas are accessed and managed by the six rings residing in the packet buffer ring memory and listed in Table 1. Note that the division is considered virtual because the physical memory locations of the areas change over time.

The three pipelined stages and the memory refresh circuitry use the packet memory interface (PMI) to access PBM. The PMI arbitrates and prioritizes the requests for memory access from these four requesters. Physically, the PMI has three interfaces: the FCP stage, the REM stage, and the HPD stage. Virtually, the PMI has four interfaces; the HPD interface multiplexes traffic from both the host and the adapter manager. The PMI also has the functionality to refresh the dynamic memory and to implement a synchronizer between the 80-nanosecond FDDI clock and the 64-nanosecond XMI clock.

All interfaces request access to the memory by invoking a request/grant protocol. Some accesses are longword (4-byte) transactions that require one to two memory cycles; others are hexaword (32-byte) transactions and require a burst of memory cycles.

The interfaces have the following priorities: (1) refresh memory circuitry, (2) the REM stage, (3) the FCP stage, and (4) the HPD stage. The refresh memory circuitry has the highest priority because data loss in the dynamic memory is disastrous. Also the refresh circuitry makes a request once every 5 to 10 microseconds, thus ensuring that the lower priority requesters always have access to the memory. The REM has the second highest priority because it always requests one longword, which requires one memory cycle. Once the REM receives data, by design it waits at least two cycles before making the next request. Thus, the REM does not monopolize the memory, and the FCP can always get its requests serviced. The FCP stage requires guaranteed memory bandwidth with small latency to avoid an overflow or underflow condition in its FIFOs. Finally, the HPD interface has the lowest priority because no data loss occurs if memory access is denied for a theoretically infinite amount of time. Our adapter design has mechanisms that guarantee memory access to the HPD.

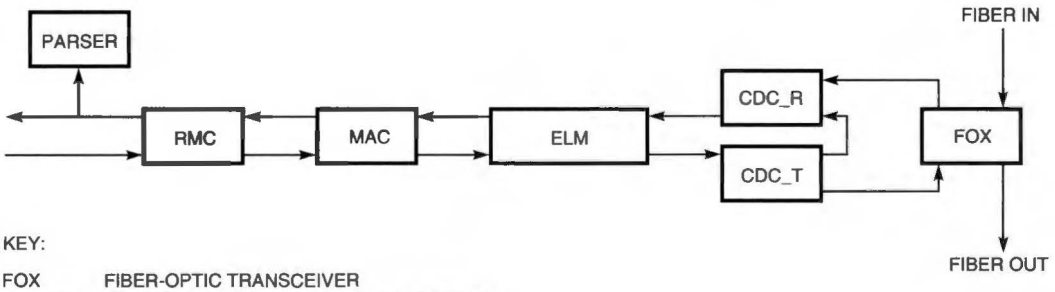
FDDI Corner and Parser Stage

The FCP stage, illustrated in Figure 5, provides the interface between the FDDI ring and the packet buffer memory. This stage can receive or transmit the smallest packet in 2.24 microseconds, as required by our performance constraints.

The receive stream in this stage converts the incoming stream of photons from the FDDI ring into a serial bit stream using the fiber-optic transceiver (FOX) chip. The clock and data conversion chip then recovers the clock and converts the incoming code from 5 to 4 bits. The MAC chip converts this electronic serial bit stream to a byte stream. The MAC chip implements a superset of the ANSI MAC standard.[9] Digital has a specific implementation of the MAC chip.[3] The ring memory controller (RMC) interfaces with the byte-wide stream from the MAC, converts the bytes into 32-bit words, and writes these words to the PBM, using the RMC receive ring and the ring protocol.

The transmit stream accesses a packet from the PBM, waits for the token on the FDDI ring, and transmits the packet as a stream of photons. This stage can generate and append 16 bytes of cyclic redundancy code (CRC) to every packet before transmitting.

The parser component of this stage interfaces with the RMC bus to generate a forwarding vector that has a variety of information including the data link user identity and the destination of the packet,



KEY:

FOX	FIBER-OPTIC TRANSCEIVER
CDC_R	CLOCK AND DATA CONVERSION RECEIVER
CDC_T	CLOCK AND DATA CONVERSION TRANSMITTER
ELM	ELASTICITY BUFFER AND LINK MANAGEMENT
MAC	MEDIA ACCESS CONTROL
RMC	RING MEMORY CONTROLLER

Figure 5 FDDI Corner and Parser Stage

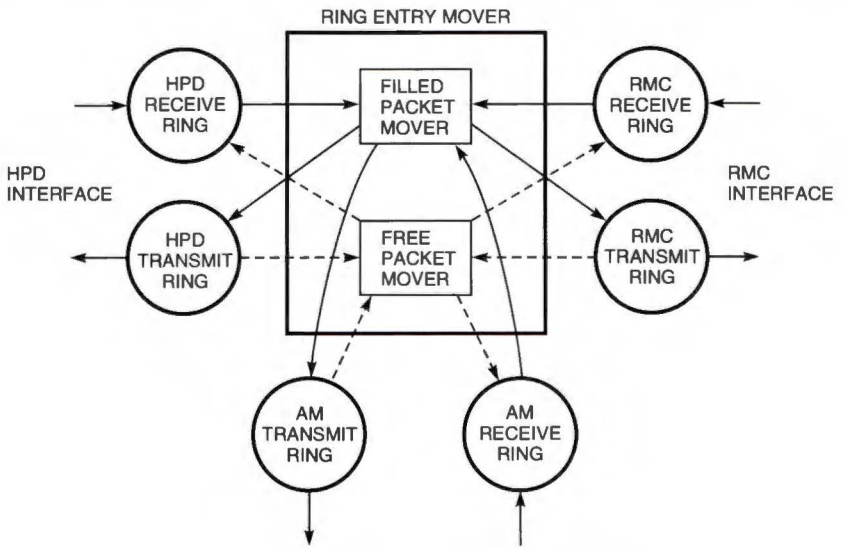
i.e., the host or the AM. The parser extracts packet headers from the RMC bus and operates on the FDDI and the LLC parts of the packet headers. The parser then processes this information in real time, using a content-addressable memory (CAM) that stores the profiles of data link and other users. As a result, the parser generates a forwarding vector that contains the destination address of either the host user or the AM user. The forwarding vector destination field is given a "discard" value, if the packet header does not match any user profile. Note that the forwarding vector is a part of the buffer descriptor field in the RMC receive ring.

Ring Entry Mover Stage

The ring entry mover stage performs four major functions: (1) moving filled packets from receive rings to transmit rings, (2) returning free packets from transmit rings to receive rings, (3) managing buffers, and (4) collecting statistics. Figure 6 shows the various rings, the ring entry mover, and the movement of filled and free packets.

The REM moves filled packets from receive rings to transmit rings by copying pointers rather than copying data. (Copying pointers is a much faster operation than data copy.) Note in Figure 6 that for a given interface, no filled packet moves from its receive ring to its transmit ring. For example, no filled packet moves from the RMC receive ring to the RMC transmit ring. Also, in this design there is no need for a path from the HPD receive ring to the AM transmit ring.

A second function performed by the REM stage is to return free packets from the transmit rings to the proper receive rings. Transmit rings point to free packets after the receiver interface has consumed



FROM \ TO	RMC TRANSMIT RING	HPD TRANSMIT RING	AM TRANSMIT RING
RMC RECEIVE RING	NO	YES	YES
HPD RECEIVE RING	YES	NO	NO (BY DESIGN)
AM RECEIVE RING	YES	YES	NO

KEY:

HPD	HOST PROTOCOL DECODER
RMC	RING MEMORY CONTROLLER
AM	ADAPTER MANAGER

Figure 6 Movement of Filled Packets by the Ring Entry Mover

the information in the packet. The REM, which is a transmitter interface on all transmit rings in the PBM, owns these buffers after the appropriate receiver interface toggles the ownership bit. The REM returns the buffers to the original receive ring by using information in the color field, a subset of the buffer descriptor field. The color field contains color information that designates the receive ring to which the buffers belong. This color information is written into the buffer descriptors of the free buffers during initialization. Note that during initialization, the adapter free buffers in the PBM are allocated to the three receive rings with which the REM interfaces.

The REM also performs buffer resource management. Note that a reserved pool of buffers exists for traffic arriving over the FDDI ring. This FDDI traffic has two destinations, namely the host CPU and the adapter manager. To ensure that one destination does not monopolize the pool of buffers, the pool is divided into two parts: host allocation and AM allocation. The REM delivers no more than the allocated number of buffers to one destination.

The fourth major function that the REM performs is to collect statistics. The REM collects statistics in discard counters for packets that cannot be delivered due to lack of resources. The REM interrupts the AM when these counters are half full. The AM reads, processes, and stores these counters for statistical purposes. The AM read operation resets these counters. There are a number of other counters in REM.

Host Protocol Decoder Stage

The host protocol decoder interfaces with the XMI bus, fetches and interprets entries from the host receive and transmit rings, and moves data between the host and the PBM. This stage also acts as a gateway for the AM to get to the host memory or to the PBM.

Figure 7 is a block diagram of the HPD stage. The receive and transmit pipelines store and retrieve receive and transmit data from the host memory. The two pipelines work in parallel. We now explain the operation of the receive pipeline in detail. The transmit pipeline operates in a similar manner; thus, we highlight only the differences.

HPD Receive Pipeline The receive pipeline has three stages: (1) the fetch and decode host receive entry stage, (2) the data mover stage, and (3) the receive buffer descriptor write stage. Most pipelines work in a lockstep fashion; that is, each stage takes the same amount of time to process input. In our design, the processing time varies for each stage in the pipeline. For example the data mover stage will take a much longer time to transfer 4500-byte packets than to transfer 100-byte packets. The fetch and decode host receive entry

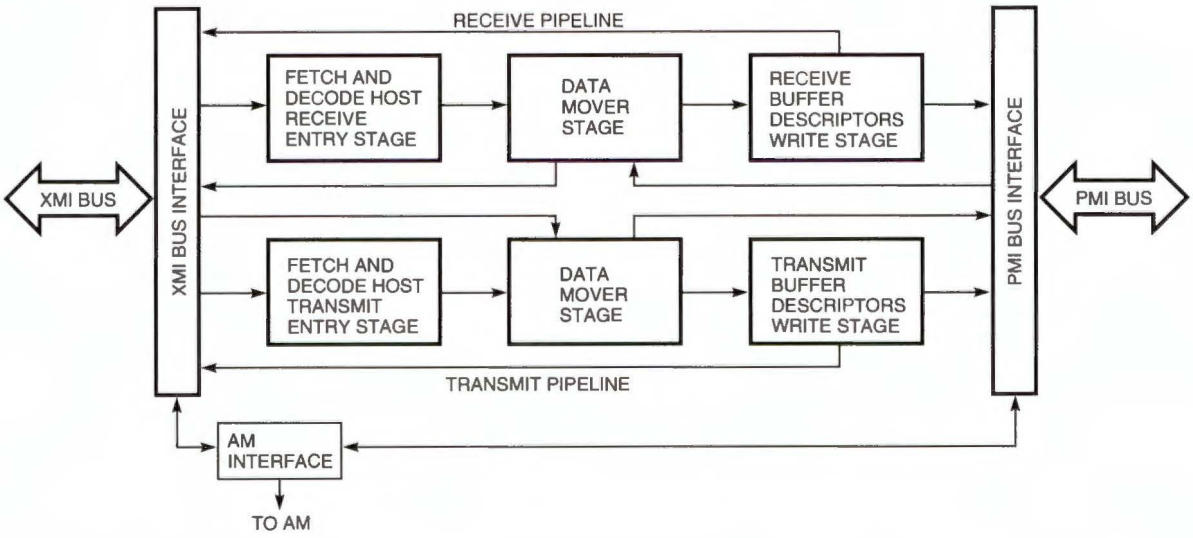


Figure 7 Host Protocol Decoder Stage

stage, on the other hand, may take the same amount of time to decode entries for packets of either size. Consequently, stages use interlocks to signal the completion of work.

The fetch and decode host receive entry stage has knowledge of the format and size of the ring and sequentially fetches host receive ring entries. If the adapter does not own an entry, this stage waits for a signal from the host before fetching the entry again. If the adapter does own the entry, this stage decodes the entry to determine the address of the free buffer in the host memory and the number of bytes in the buffer. The stage then passes this buffer information to the data mover stage and the address of the host entry to the receive buffer descriptor write stage. In addition, this stage prefetches the next entry to keep the pipeline full, in case data is actively received over the FDDI ring.

In parallel, the PMI interface stage part of the HPD chip fetches the next entry from the HPD transmit ring. Decoding this entry determines the address of the buffer in the PBM and the amount of data in the buffer. The packet buffer bus interface passes the buffer address and length information to the data mover stage and the address of the HPD transmit ring entry to the receive buffer descriptor write stage.

Now, the data mover stage has pointers to the host free buffer and its extent and to the PBM filled buffer and its extent. The stage proceeds to move the data from the PBM to the host memory over the XMI bus. Depending on the XMI memory design, this transfer involves octaword or hexaword bursts. The process of moving data continues until the depletion of packet data in the PBM.

The data mover stage signals the receive buffer descriptor stage when the packet moving is complete. The receive buffer descriptor stage writes in the status fields of the host receive ring entry and the HPD transmit ring entry. This stage also gives ownership of the filled buffer to the host and of the free buffer to the REM. The REM can then return the free buffer to the ring of origin.

HPD Transmit Pipeline The HPD transmit and receive pipelines are symmetrical. The HPD receive pipeline delivers data from the HPD transmit ring to the host receive ring. The HPD transmit pipeline delivers data from the host transmit ring to the HPD receive ring.

There is one exception to the symmetry. The transmit pipeline does not fetch an entry from the HPD receive ring in PBM to determine if there are enough free buffers available. A hardware interface between the PMI and the HPD, i.e., a Boolean signal, indicates whether there are enough buffers to accommodate the largest possible size transmit packet. This exception is an artifact of our implementation; we wanted to reduce the accesses to the PBM, since its bandwidth is a scarce resource.

Adapter Manager

The local intelligence, also known as the adapter manager, implements various necessary adapter functions including self-test and the initialization. The AM also implements part of the CMT code that manages the FDDI connection.[10] In addition, the AM interfaces with the host to start and stop data link users by dynamically manipulating the parser data base.

Tracing a Packet through the Adapter

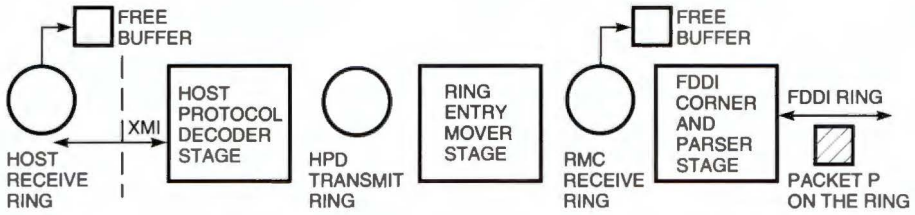
The major steps for data transfer incorporate the subfunctions previously discussed. This section traces the path of a packet P through the adapter, first on the receive stream and then on the transmit stream. We assume that adapter initialization is complete and that all data structures in the packet memory and parser data base are properly set. In this example, we further assume that packet P is small enough to fit into a single buffer. Large packets require multiple buffers.

Receive Stream

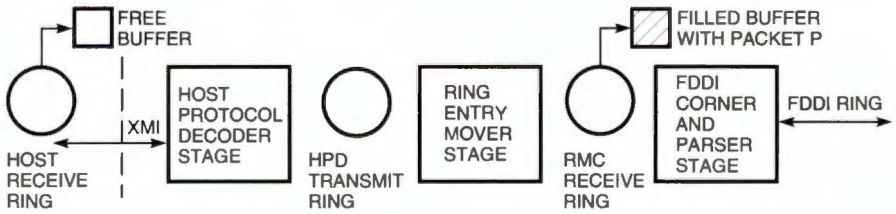
A packet destined for the host passes through the three major pipelined stages in the adapter. A brief description of the intrastage operation and details of the interstage functioning follow. The four parts of Figure 8 illustrate the receive process.

FDDI Corner and Parser Stage Figure 8(a) shows packet P on the FDDI ring; the packet is actually a stream of photons. This stage converts the stream of photons into a packet. At this point, a free buffer is available for packet P in both the RMC receive ring and the host receive ring. The FCP stage owns the free buffer in the RMC receive ring.

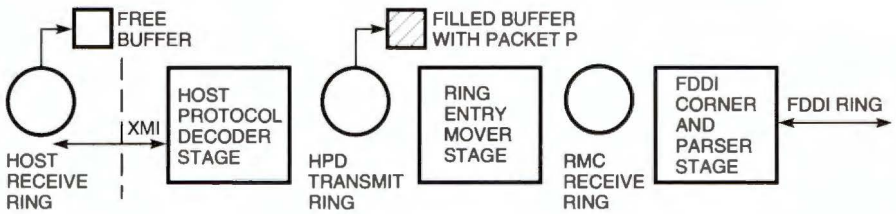
The stage determines if packet P is addressed to this node, forwards the packet on the FDDI ring, and copies the packet for this adapter if it is addressed to this node. This stage also generates a CRC for the packet. The FCP stage then deposits the copied packet into the free buffer in the RMC receive ring entry shown in Figure 8(b).



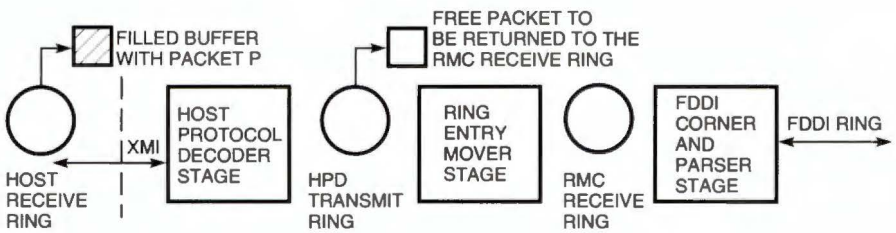
(a) Receive Stream - Packet on the FDDI Ring



(b) Receive Stream - Packet at the RMC Receive Ring



(c) Receive Stream - Packet at the HPD Transmit Ring



(d) Receive Stream - Packet in the Host Memory on the Host Receive Ring

Figure 8 Receive Stream—The Receipt of a Packet from the FDDI Ring to the Host Memory

After depositing the complete packet, this stage writes the buffer descriptor and toggles the ownership bit. The ring entry mover now owns packet P. The FCP stage is free to receive the next packet, which is stored in the next buffer in the RMC receive ring.

Ring Entry Mover Stage The REM extracts the packet buffer descriptor and determines the number of pages in packet P. This stage also has an account of the number of pages outstanding on the HPD transmit ring. The REM delivers packet P to the HPD transmit ring provided the host resource allocation is not exceeded.

The REM delivers the packet by copying page pointers from the RMC receive ring to the HPD transmit ring, as shown in Figure 8(c). Note that the HPD transmit ring is large enough to write all

pointers from the RMC receive ring and the AM receive ring. The REM then transfers ownership of the HPD transmit ring entry to the HPD stage and the RMC receive ring entries to the FCP stage.

HPD Stage The HPD receive pipeline operates on a packet it owns in the HPD transmit ring. As shown in Figure 8(d), after fetching the address of the free host buffer, this pipeline moves packet P from the PBM to the host memory and toggles the ownership bit of the host entry. Simultaneously, the HPD returns ownership of the free buffers in the HPD transmit ring to the ring entry mover stage. The REM returns these buffers to the RMC receive ring as free buffers.

Transmit Stream

To transmit data from the host transmit ring to the FDDI ring, the packet must pass through the same three stages as for the receive stream, but in the reverse direction.

HPD Stage For the receive stream, the HPD receive pipeline prefetches the free buffer from the host receive ring. In contrast, the HPD transmit pipeline must wait for the host to fill the transmit buffer and transfer ownership to the host transmit ring. The HPD stage then moves the data from the host memory to the PBM if the hardwired signal between the REM and the HPD indicates that a sufficient number of pages is available. Finally, the HPD transfers ownership of the host transmit ring entry to the host and the HPD receive ring entry to the REM.

Ring Entry Mover Stage The REM moves the packet from the HPD receive ring to the RMC transmit ring. Again, the REM copies pointers from ring to ring and toggles the ownership bit on the RMC transmit ring.

FDDI Corner and Parser Stage Although the packet is available in PBM for transmission, the FCP stage must receive a token before transmitting over the FDDI ring. Once the transmission is complete, the buffer on the RMC transmit ring is now free. The FCP stage returns ownership of the buffer to the REM, which then returns the free buffer back to the HPD receive ring or the AM receive ring, depending upon the origin. Again, the free buffers are returned by copying buffer pointers.

The receive and transmit streams for the adapter manager are similar to those for the host; therefore, we do not describe these processes.

Hardware and Firmware Implementation

The hardware implementation of DEMFA consisted of four large gate arrays, custom very large-scale integration (VLSI) chips, dynamic and static random access memories (RAMs), and jelly bean logic. Figure 9 is a photograph of the DEMFA board.

The four gate arrays specified and designed by the group are the parser, the adapter manager interface (AMI), the host protocol decoder, and the packet memory controller (PMC), which incorporated the function of the packet memory interface and the ring memory controller. We now describe aspects of the gate array development. Note that we used the Compacted Array technology developed using LSI logic for our implementation. The gate arrays have 224-pin surface mount packaging.

Table 2 shows various gate arrays, the total gate count for each gate array, and the percentage of control gates and data path gates. Control gates are defined as gates required for implementing state machines used for control. Data path gates are gates required for registers and multiplexors, for example. Note that the complexity of gate arrays is proportional to the percentage of control gates. The gate arrays in Table 2 were fairly complex because they consisted of approximately 50 percent control gates.

Module Implementation

We used the 11-by-9-inch XMI module for implementing the adapter. Early in the project we defined the pin functions for various gate arrays. Once these were defined we could design our module. SPICE modeling helped in arriving at a correct module design with the first fabrication. The design was thorough and completed early in the project.

Firmware Implementation

The DEMFA firmware has three major functions: self-test, FDDI management (using Common Node Software), and adapter functional firmware. The DEMFA team implemented the adapter functional firmware while other groups designed the two remaining components. The DEMFA functional firmware can initialize the adapter and then interact with the host to start and stop data link layer users, as well as perform other functions. The firmware is implemented in the C language for the Motorola 68020 system. The total image size is approximately 160 kilobytes.

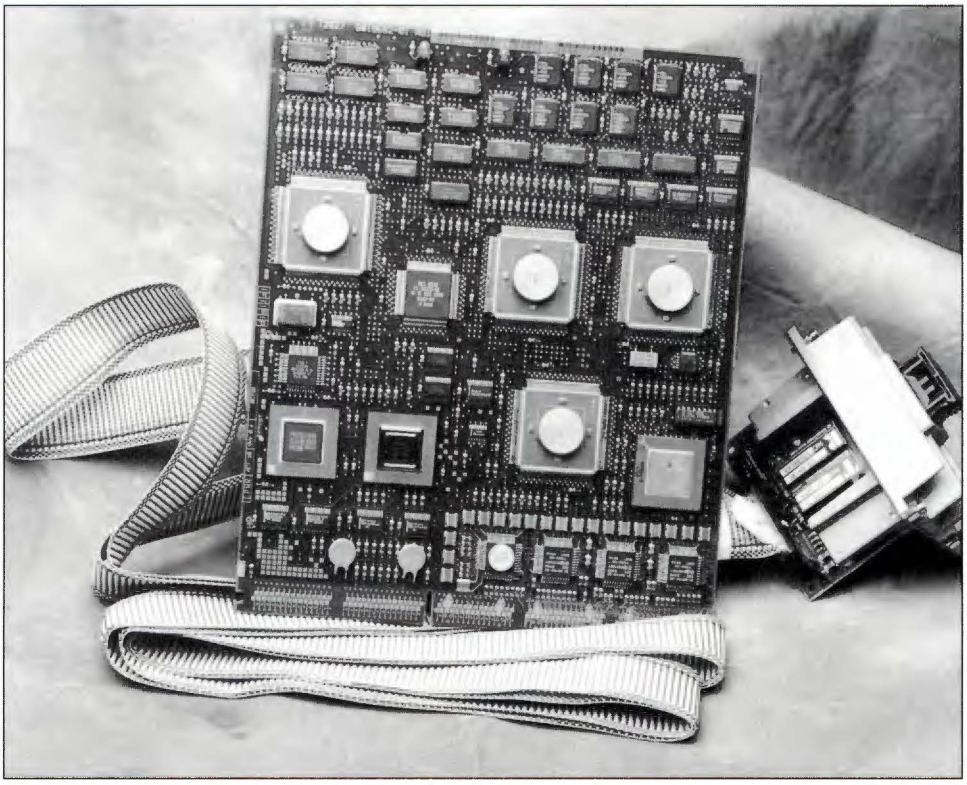


Figure 9 The DEMFA Board

Table 2 Gate Counts for DEMFA Gate Arrays

Gate Array	Total Gates	Data Gates (Percent of total)	Control Gates (Percent of total)
Parser	20296	39	61
PMC	61537	40	60
HPD	81265	34	66
AMI	15002	49	51

Performance

The graph presented in Figure 10 shows the adapter performance for the receive and transmit streams at the adapter hardware level for this implementation. The data represents throughput measured in megabits per second as a function of packet size measured in bytes. Figure 10 illustrates that the receive and transmit streams meet the 100-Mb/s throughput when the packet size is approximately 69 bytes. The bottlenecks in this implementation of the DEMFA architecture are (1) the PMI and (2) the combination of the XMI interface, bus, and memory. We implemented these interfaces in a conservative manner to reduce our risks and to produce the product in a timely fashion.

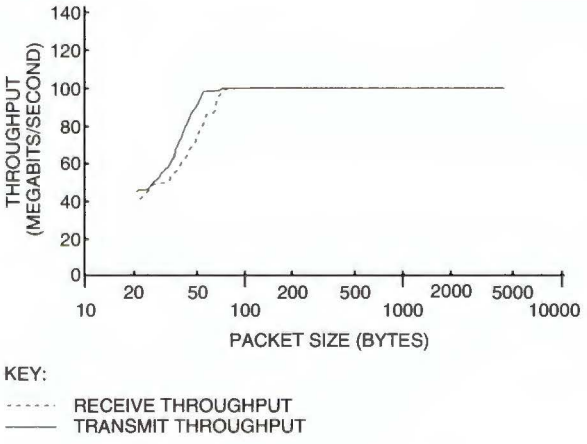


Figure 10 Adapter Performance

For more detailed performance data, see the paper entitled "Performance Analysis of a High-speed FDDI Adapter" in this issue of the *Digital Technical Journal.*[11]

Conclusion

The goal of the DEMFA project was to specify an architecture for an adapter that would be at least 30 times faster than any previously built adapter. The architecture also had to be easy to implement. This paper describes the architecture and an implementation of DEMFA. Performance measurements of the adapter show that this first implementation successfully meets close to the maximum FDDI throughput capacity; thus, the DEMFA performance can be considered ultimate. Already, a number of adapters have been designed based on ideas borrowed from the DEMFA architecture and implementation. In a few years, architectures similar to this one may become the norm for data link and even transport layer adapters, rather than the exception.

Acknowledgments

I wish to acknowledge and thank my manager, Howard Hayakawa, who out of nowhere presented me with the challenge of defining an architecture and an implementation for an FDDI adapter that would have a performance 30 times that of any existing adapter. I must have taken leave of my senses to take on such a challenge. But the end results were worth the effort.

I would also like to thank Gerard Koeckhoven, who agreed to be the engineering manager for this adapter project. He took on the consequent challenge and the risk and supported me all along the way. In addition, I want to recognize Mark Kempf for the many hours he spent helping us during the conceptualization period and for chiseling our design. The TAN architects were of great assistance in making sure that our adapter met the FDDI standard.

I also wish to acknowledge the following members of the DEMFA project team for their contributions: Santosh Hasani and Ken Wong, who designed the parser gate array; Dave Valley and Dominic Gasbarro, who designed the AMI gate array; Andy Russo and John Bridge, who designed the HPD gate array; Ron Edgar, along with the other PMC designers, Walter Kelt, Joan Klebes, Lea Walton, and Ken Wong; Ed Wu and Bob Hommel, who designed and implemented the module; the team that implemented the functional firmware, Ed Sullivan, David Dagg, Da-Hai Ding, and Martin Griesmer; and Ram Kalkunte, for designing and building a simulation model to accurately predict the performance well before the adapter was ready. Also, I would like to thank VMS and ULTRIX group members Dave Gagne, Bill Salkewicz, Dick Stockdale, and Fred Templin.

References

1. B. Allison, "An Overview of the VAX 6200 Family of Systems," *Digital Technical Journal,* no. 7 (August 1988): 10–18.

2. D. Fite, Jr., T. Fossum, and D. Manley, "Design Strategy for the VAX 9000 System," *Digital Technical Journal,* vol. 2, no. 4 (Fall 1990): 13–24.

3. H. Yang, B. Spinney, and S. Towning, "FDDI Data Link Development," *Digital Technical Journal,* vol. 3, no. 2 (Spring 1991): 31–41.

4. A. Tanenbaum, *Computer Networks* (Englewood Cliffs, NJ: Prentice Hall, Inc., 1981).

5. B. Allison, "The Architectural Definition Process of the VAX 6200 Family," *Digital Technical Journal,* no. 7 (August 1988): 19–27.

6. *Token Ring Access Method and Physical Layer Specifications,* ANSI/IEEE Standard 802.5-1989 (New York: The Institute of Electrical and Electronics Engineers, Inc., 1989).

7. R. Stockdale and J. Weiss, "Design of the DEC LANcontroller 400 Adapter," *Digital Technical Journal,* vol. 3, no. 3 (Summer 1991, this issue): 36–47.

8. *FDDI Station Management (SMT),* Preliminary Draft, Proposed American National Standard, ANSI X3T9/90-X3T9.5/84-49, REV. 6.2 (May 1990).

9. *Token Ring Media Access Control (MAC),* (International Standards Organization, reference no. ISO 9314-2, 1989).

10. P. Ciarfella, D. Benson, and D. Sawyer, "An Overview of the Common Node Software," *Digital Technical Journal,* vol. 3, no. 2 (Spring 1991): 42–52.

11. R. Kalkunte, "Performance Analysis of a High-speed FDDI Adapter," *Digital Technical Journal,* vol.3, no.3 (Summer 1991, this issue): 64–77.

Ramsesh S. Kalkunte

Performance Analysis of a High-speed FDDI Adapter

The DEC FDDIcontroller 400 host-to-FDDI network adapter implements real-time processing functionality in hardware, unlike conventional microprocessor-based designs. To develop this high-performance product with the available technological resources and at minimal cost, we optimized the adapter design by creating a simulation model. This model, apart from predicting performance, enabled engineers to analyze the functional correctness and the performance impact of potential designs. As a result, our implementation delivers close to ultimate performance for an FDDI adapter and surpasses the initial project expectations.

As high-performance systems become available and the use of distributed computing proliferates, the need for high-performance networks increases. Faster interconnects are required to achieve such performance goals. Consequently, network adapters must be able to function at higher speeds. In adopting fiber distributed data interface (FDDI) local area network (LAN) technology as a follow-on to Ethernet, Digital recognized the need to build an industry-leading network adapter to service its high-performance platforms. As a result, we designed and developed the DEC FDDIcontroller 400 product. To track the adapter performance through the design and development stages, we created a simulation model; our objective was to ensure that the device met our performance goals. This paper begins with a description of the DEC FDDIcontroller 400, followed by a brief historical perspective and statement of the performance objectives of the adapter project. We then discuss in detail the modeling methodology and the results achieved. In addition, we present validation of these results in the form of measurements taken on prototype hardware.

The DEC FDDIcontroller 400

The DEC FDDIcontroller 400, also known as the DEMFA, is a high-speed FDDI network adapter. Attached to a host machine running under either the VMS or the ULTRIX operating system, the DEMFA enables the host to communicate with other network entities through the FDDI ring. The DEMFA adapter implements Digital's proprietary XMI bus protocol and can be used with any system that has an XMI backplane.[1] Laboratory measured performance data presented later in the paper shows that the adapter hardware can sustain a practically infinite stream of frames at the full FDDI data bandwidth of 100 megabits per second (Mb/s) for frame sizes 69 bytes or larger on the receive stream and 51 bytes or larger on the transmit stream. Even the smallest, i.e., 20-byte dataless, FDDI frames can be received at 36 Mb/s and transmitted at 47 Mb/s.

The DEMFA is an FDDI Class-B single attachment station (SAS) that interfaces to the FDDI token ring network through the DECconcentrator 500. A port driver resident in the host controls the DEMFA port. The port, the port driver, and the adapter hardware implement the American National Standards Institute (ANSI) data link and physical layer functionality for FDDI LANs. This foundation supports user protocols such as the Open Systems Interconnection (OSI), DECnet, the transmission control protocol with the internet protocol (TCP/IP), and local area transport (LAT).[2] Figure 1 shows a typical network configuration using the DEC FDDIcontroller 400 adapter with other Digital FDDI products.

The XMI bus is capable of transferring data at rates up to 800 Mb/s and can serve as either a CPU-to-memory interconnect, e.g., in the VAX 6000 platform, or an I/O bus, e.g., in the VAX 9000 platform.[3,4] Also, Digital plans to include the XMI bus in future systems.

FDDI is a timed-token, fiber-optic ring that provides a network data bandwidth of 100 Mb/s.[5] In addition to this high data rate, the advantages of low signal

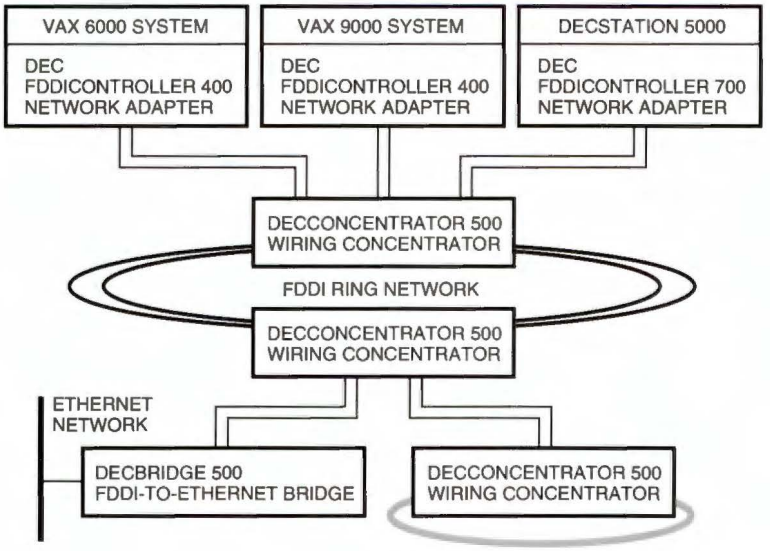


Figure 1 Typical Network Configuration

attenuation, low noise susceptibility, high security, and low cost (as the technology matures) will make FDDI a popular interconnect of the 1990s.[6]

Historical Perspective and Performance Objectives of the DEMFA

With the advent of high-performance systems and distributed computing strategies, the need for high-performance networking options has increased. Traditionally, I/O adapters have been built to serve the current performance needs. As a consequence, such adapters offer little or no network performance scalability to accommodate future increases in demand. Scalability is important to ensure that the adapter does not become a bottleneck when such demands exist. Nonscalable adapters become obsolete, and the resulting frequent hardware upgrades increase system cost.

The first Ethernet adapters, which complied with the IEEE 802.3 standard, were built in the early 1980s. Only recently do adapters exist that can process frames at the maximum Ethernet throughput rate of 10 Mb/s.[7] As mentioned earlier, FDDI has the capability of supporting speeds an order of magnitude higher than Ethernet. Since the header in an FDDI frame is three times smaller than that for Ethernet, FDDI frame arrival rates can be as much as 30 times the Ethernet arrival rate. Considering the various constraints, Digital set out with the goal to build an FDDI adapter that could process frames 150 bytes and larger at 100 Mb/s, i.e., the adapter would be able to process approximately 80,000 frames per second (frames/s). Also, twenty microseconds was deemed an acceptable adapter latency for the smallest FDDI frames. Considering the relatively small number of frames a host system can process today, these adapter criteria represented an ambitious goal—one which would make a product with high-performance scalability as faster CPUs became available.

Performance Modeling Considerations

During the development of a high-performance product, changes in architectural functionality, technology constraints, and cost considerations can result in design modifications. It is desirable to track the performance of the product through its development to understand the impact of such modifications.

The DEMFA consists of many hardware entities that perform the desired adapter functions.[8] Although such hardware adapters have the obvious advantage of superior performance over conventional, i.e., microprocessor-based adapter cards, this advantage does not come without the risks associated with hardwired logic. Such risks have a negative impact on project budget and schedule and necessitate a risk management strategy to ensure that product goals are successfully met. Performance modeling of the adapter and extending the use of such modeling to evaluate various designs formed part of this strategy. The following subsections describe the goals and tasks of the DEMFA performance modeling.

Goals

The set of performance modeling goals for the DEMFA evolved throughout the development process. Three major goals were performance projection, buffer sufficiency analysis, and design testing through simulation.

Performance Projection In the early phases of the design, the primary goal of the model was to project the adapter performance. This prediction gave us confidence that the design could meet our performance expectations.

Buffer Sufficiency Analysis Buffer capacity plays an important part in the performance of a design. Whereas too much of this resource is wasteful, too little has a negative effect on performance. It was critical to determine the extent of buffering necessary to attain the desired target performance at the least cost. The performance model considered the dependencies on this resource. The amount of buffering was varied and the effects of such variation, manifested in the simulation results, were analyzed. Using these results as input to a cost/benefits equation helped the designers make intelligent decisions concerning buffer capacity.

Design Testing through Simulation As development progressed, important design issues arose that could not be solved by simple analysis. The performance model served as a platform that could be enhanced to solve these more complex problems by simulation. Designs were analyzed to determine their impact on adapter performance. Because the simulation methodology afforded greater testability, we were able to make the designs more robust and to answer design questions in a significantly shorter time than other methods. Consequently, modifications to the hardware were made at an early design stage and at negligible cost.

Tasks

To accomplish performance modeling, we faced the following basic tasks: choosing the metrics, defining the workload, and deciding on a modeling methodology. Relevant metrics to measure the performance of a product are crucial. We chose metrics that are simple to understand and provide insight into the behavior of the product. Also, areas in which workload development is required must be identified and investigated in detail. An incorrect workload invalidates all performance data. And the methodology used to model the system must be well thought-out beforehand, so that the model is accurate and also flexible enough to be easily changed.

Definition of Metrics The main performance metrics used were throughput and frame latency. Throughput is the rate at which frames are processed and is measured in megabits per second or frames per second. The units can be converted easily from one to the other, if the average frame size is specified. In this paper, throughput is expressed in megabits per second.

Frame latency is the elapsed time measured in microseconds between the time at which a frame is queued for service at a facility and the time at which the service is completed. The following descriptions illustrate the approach used to measure receive and transmit latency. The host receives frames from and transmits frames to the FDDI ring. Receive frame latency is the time elapsed between (1) the arrival of the last bit of the frame into the adapter from the FDDI ring and (2) the time the frame becomes available to the host for processing. Transmit frame latency is the elapsed time between (1) the time the adapter starts processing a frame from the host and (2) the exit time of the first bit of the frame from the adapter destined for the FDDI ring.

The adapter can process transmit and receive frames simultaneously. We defined performance metrics to analyze a variety of traffic scenarios to gain insight into the adapter behavior. For the context of this paper, we consider the DEMFA processing pure frame streams only, i.e., the expressions "receive throughput" and "receive latency" refer to a pure receive stream of frames containing no transmit frames. Similarly, "transmit throughput" and "transmit latency" refer to a pure transmit stream of frames.

Workload Definition Using a relevant traffic workload is very important in any simulation model. Since most systems are workload-sensitive, defining an incorrect workload may result in irrelevant data. We identified two areas in which we needed to define workloads. We then characterized the traffic patterns and built a workload model for performance simulation based on these patterns.

- Frame receive and transmit workloads. The receive and transmit workloads are stimuli for the performance simulation. These workloads

mimic traffic due to frame arrival on the FDDI ring (i.e., the receive workload) or frame transmission from the host (i.e., the transmit workload). The receive workload model generates frames which the DEMFA model receives, whereas the transmit workload acts as a source of frames to be transmitted by the DEMFA model on the FDDI ring. These workloads must be characteristic of actual FDDI traffic. Since FDDI LANs did not exist when the DEMFA was in the development stage, we used our experiences with Ethernet to derive these workloads, as we explain in greater detail in the FDDI Token Ring section.

- XMI traffic workload. Apart from the DEMFA traffic, there may be other traffic on the XMI bus due to CPU-to-memory transactions or from other I/O adapters attached to the system. The load on the XMI bus impacts the performance of the DEMFA. Consequently, we designed a workload model to mimic the traffic pattern on the bus. We based our model on the traffic patterns observed for real XMI bus traffic. The performance of DEMFA may degrade as this traffic increases because the DEMFA traffic and the non-DEMFA traffic consume common resources. The other traffic is referred to as the XMI interference workload. The XMI Workload Generator section describes the model for this workload.

Modeling Methodology The simulation model has a hierarchical design to allow the construction of smaller, more manageable blocks, i.e., submodels. The structure also allows changes to be made easily. The SIMULA language implements the simulation model.[9] The simulation-class and queuing constructs in this language are tailored to help simulation and modeling.[10,11] The object-oriented structures present other advantages to model development. A debug procedure coded into the model prints status information about all the queues in the model. This information helped us trace the path of frames through the system.

One important first step in designing a simulation model is to determine the detail at which to model. Two factors that influence the level of detail are the

- Existing knowledge of the design. Usually, information gathered from the behavioral and analytical models of a design helps to make a performance model abstraction. Designs with behavior that cannot be analyzed by these lower-level models have to be modeled in greater detail.
- Expectation of performance model accuracy. Typically, a performance model predicts results accurate to within ± 10.0 percent of the performance that would be achieved with the actual hardware.

During the design phase, behavioral and structural models of hardware were in development. This hardware was partitioned across important functional boundaries. Hardware within these boundaries would be modeled and tested thoroughly by the respective development engineers. Hence, to include details of these pieces of hardware in our model would have resulted in redundant effort. Since the interfaces and the gross functionality of the hardware within these boundaries are relevant to performance, we did include these components in our model. Existing hardware components, such as the FDDI chip set, were grouped together before being modeled for functionality. Each submodel was designed and tested separately to ensure conformity to the functionality and performance of other behavioral and structural models. This strategy resulted in the base-level performance model that we used to generate preliminary performance data for the DEMFA.

As development progressed, we encountered design changes of various complexities. Simple design changes resulted in very small changes in the performance model. But larger and more complex design changes required that we investigate behavior both specific to the piece of hardware of which the design is a part and generalized to the adapter system environment in which the piece operates. Models that represent the changes were included and interfaced as submodels. These submodels served the dual purposes of testing the new design and of improving the accuracy of the performance model.

Design of the Simulation Model

The performance simulation model consisted of the following major components:

- FDDI ring
- FDDI chip set and parser
- Packet memory controller
- Host interface
- XMI system
- Host system

The base-level model evolved over time, as we gained insight into the behavior of the individual components and defined workloads. The model evolved further to support the need to analyze new designs through simulation. This section briefly describes the components of the final model, as listed above.

FDDI Token Ring

The FDDI token ring was modeled to act as a source of received frames and as a sink of transmit frames. Gross functionality for the remainder of the FDDI nodes and network components was desirable. Consequently, we designed a black-box model for the FDDI ring that provides two-way interaction with the FDDI chip set and parser model. This FDDI model allocates time on the FDDI ring for transmit and receive transactions. The model also controls a receive workload generator when frames are received by the adapter.

The receive workload generator is an analytical model used to create different patterns of receive traffic to the DEMFA. The parameters input to this workload model are the average frame size, the frame-size distribution, the frame type, the load, and the number of back-to-back frame arrivals (i.e., the burst rate or "burstiness" of the frame arrivals). We varied these parameters to generate desired workloads.

The average frame size and frame-size distribution parameters generate different size frames. Actual frame sizes can be specified as normally or exponentially distributed about the mean or as constant. The workload model can generate station management (SMT), LLC SNAP/SAP, or LLC non-SNAP/SAP frame types and can create a load between 0 and 100 Mb/s. If workloads are less than the peak FDDI bandwidth, i.e., 100 Mb/s, the frame arrival pattern can be specified as an exponential, constant, or normal distribution. The model can generate a wide range of synthetic traffic patterns, but to obtain credible performance results, we characterized the traffic as seen in realistic networks.

Several studies had been conducted on large Ethernet LANs within Digital; a case study by D. Chiu and R. Sudama is one example.[12] We analyzed the results from these studies to understand the frame-size distribution in such networks. From the analysis we concluded that

- Frame sizes on the networks are related to user protocols. Frames in a test sample were distributed about a few discrete frame sizes (i.e., modes of the distribution) rather than over a wide range of frame sizes.
- The probability function of the frame sizes near each mode can be approximated as a normal distribution centered about the mode.

A composition analysis of the measurements provided different modal mean sizes, standard deviations, and the probabilities of frames belonging to the different modes. We used these values to statistically create Ethernet network traffic. For our performance measurements, it was necessary for us to change this traffic pattern appropriately to reflect the differences that exist between FDDI LANs and Ethernet LANs. The FDDI frame header is smaller than the Ethernet header, and the largest FDDI frame is approximately three times the size of the largest Ethernet frame. We factored these changes into the Ethernet model to produce an FDDI workload model. The FDDI workload has either four or five modes.

The four-mode distribution contained a majority of frames grouped around 60, 576, 1518, and 4496 bytes. The standard deviations of the frames around these mean values were 22, 5, 2, and 2 bytes, respectively. The frame volumes at these modal values represented contributions of 29 percent, 67 percent, 3 percent, and 1 percent, respectively, to the total load.

The five-mode frame sizes were grouped around 33, 80, 576, 1518, and 4496 bytes. The standard deviations of the frames around these means were 1, 20, 5, 2, and 2, respectively. The frame volumes at these modes contributed 26 percent, 15 percent, 55 percent, 3 percent, and 1 percent, respectively, to the total load.

In the above FDDI workload model, the mode of 1518 bytes is determined by the Ethernet network's maximum frame-size capacity and, similarly, the mode of 4496 bytes is determined by the FDDI network's maximum frame-size capacity. These two modal frame sizes represent traffic generated by large data transfer operations, e.g., file transfers. Contributions due to these two modes vary from network to network. We considered different contributions and found their effect on adapter throughput to be negligible. Therefore, only one case for each workload is presented in this paper.

FDDI Chip Set and Parser

The FDDI chip set, also referred to as the FDDI corner, is the base-level technology that was part

of Digital's strategy to build high-performance, low-cost data links for FDDI LANs. This chip set performs serial-to-parallel data conversion, acts as an interface to the packet memory in the data link layer, and can support a data rate of 100 Mb/s.[5] The entire chip set, except for the ring memory controller (RMC), was modeled as a black box with a specified per-frame latency. The RMC and the associated first in, first out (FIFO) buffers for the receive and transmit stream staging were modeled in greater detail. The detail was necessary to capture any overflow or underflow conditions that might occur in the FIFO buffers. We also modeled the interaction between the transmit and receive streams. The RMC model, which served as the front end of the chip set model, was also capable of generating control and data transactions to perform read/write memory operations.

The parser hardware off-loads some host frame processing to the adapter. The parser reads information about a receive frame from the RMC bus and creates a forwarding vector, which is appended to the frame. This forwarding vector is used by different entities in the adapter and the host to efficiently process a frame. The parser latency to generate this vector varies with the frame type and size. The parser model helped to analyze the impact of this latency on performance. This model mimics the hardware to produce a forwarding vector for a given frame with a pertinent latency.

Packet Memory Controller

The packet memory controller (PMC) is the heart of the adapter system. The ring entry mover stage, the packet buffer memory, and the packet memory interface constitute the functionality in the PMC.[8] The PMC controls the arbitration and servicing of requests to and from memory to effect the efficient transfer of information. The PMC also controls the movement of pointers corresponding to every frame. These pointers and the associated protocol generate work for the RMC, the host interface, or the adapter manager.

The high throughput capability of FDDI rings can result in traffic patterns that cause a strain on the packet memory. The PMC model allowed us to study such scenarios. It is also important to analyze the working and performance of the ring entry mover, which moves frames between different interfaces by manipulating the control information of a stored frame. The control information and frame data reside in the packet memory.

Host Interface

The host interface, also called the host protocol decoder, moves data between the adapter and the host system through an XMI bus and also interfaces with the PMC. We modeled the interface to include details of the dual direct memory access (DMA) design (one channel for the receive stream and one for the transmit stream), the staging buffers associated with each DMA channel, the XMI interface, and the PMC interface. The host interface also has the capability of scheduling write operations while waiting for the delivery of read information. Priority schemes to complete such transactions, i.e., handshake mechanisms, are important from a performance perspective and, hence, were included in the model.

XMI System

The XMI system interacts with the host system and was modeled to include details of the XMI bus and memory. This model consists of an XMI bus submodel that interfaces to the XMI end of the host interface model of the adapter. The submodel also interacts with a memory model and an XMI workload generator model. The bus submodel implements the XMI protocol.

Memory Model The memory model was designed to generate responses to transactions that request memory. Latency for these requests is the memory access time, which includes a queue wait time. There are basically two types of systems that support the DEMFA, as shown in Figure 2. The type is determined by whether the XMI is used as the CPU bus, denoted in this paper as the XMI (CPU) bus configuration, or as the I/O bus, denoted as the XMI (I/O) bus configuration. The only difference between the two systems is memory access time. This time is greater if XMI is used as the I/O bus; there is an added latency on the read transactions performed to fetch memory from locations that are not local to the XMI bus. The memory space that is local to the CPU bus is accessed through another I/O adapter mechanism. Such I/O adapters, CPU buses, and main memory bandwidth all play a role in determining the access times in such systems. The model presented in this paper depicts the VAX 9000 I/O architecture and current implementation. Performance may vary with other implementations.

XMI Workload Generator We designed the XMI workload generator to represent the load on the

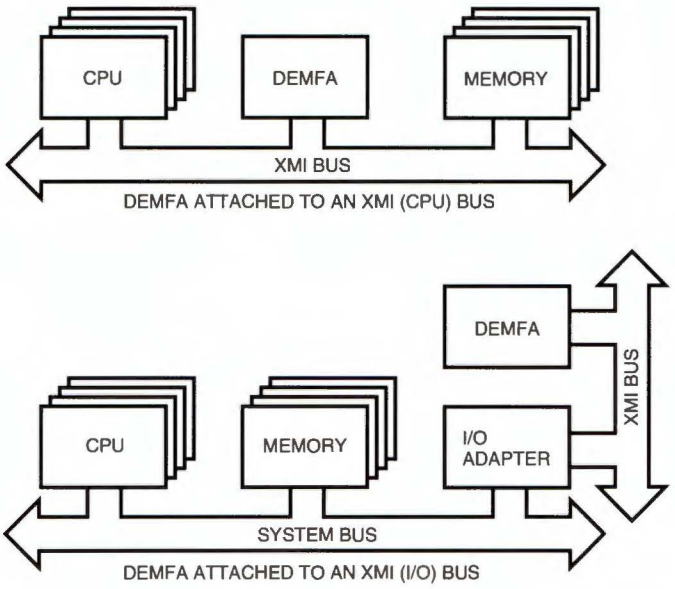


Figure 2 System Types That Support the DEMFA

XMI bus, excluding traffic from the DEMFA. This load tends to have a deteriorating effect on DEMFA performance and thus, is referred to as the XMI interference workload. It was important not only to model the amount of load but also to capture the arrival pattern of this traffic. The workload model generated traffic based on three inputs: the total XMI bandwidth used by other XMI nodes, the average length of each XMI transaction, and the burst rate of the frame arrivals. Transaction lengths on XMI vary from one to five XMI cycles (i.e., 64-nanosecond cycles). The maximum number of nodes that can exist on an XMI bus is 14. Thus, the burst rate can vary from 1 to 13.

Typically, traffic on an XMI bus consists of many back-to-back transactions of various sizes. We decided to use the worst case values for both the burst rate and the transaction length in the XMI interference workload presented in this paper. The worst case burst rate is 13, and the worst case transaction length is 5 XMI cycles.

Host System

The host system consists of the CPU, disks, layered software, the operating system, the device driver, and a host workload generator. The host system was modeled in accordance with assumptions presented in the section Results from Performance Simulation. The CPU, disks, host software, and the operating system were modeled in such a way that they do not become bottlenecks during frame reception or transmission. A model of the device driver handles frame transmission and reception. The driver interacts with a host workload generator, which creates different traffic patterns for transmission. This workload generator has the same capabilities as the receive workload generator discussed in an earlier section.

Results from Performance Simulation

The data presented in this section was generated using the simulation model of the adapter. This data represents the hardware performance of the DEMFA; system performance with the DEMFA as a component is not within the scope of this paper. We input parameters to the simulation model that defined traffic patterns and ran simulations for a sufficient length of time to ensure that we captured steady-state behavior. The models maintain statistics of relevant events and quantities and print out this information at the end of a simulation. As discussed previously, the hardware performance of the DEMFA varies depending upon whether the system is implemented to use the XMI bus as a CPU bus or as an I/O bus. This section presents simulation results for both uses, where appropriate.

Assumptions

For our simulation purposes, we made several assumptions. These assumptions make the results

more general and bring out the hardware performance characteristics of the DEMFA, indicating the upper bounds of performance that the adapter can achieve.

CPU and Software Capabilities The device driver and the host software do not become bottlenecks during frame reception and transmission. We assumed that the host CPU had enough computing ability to process frames without posing as a performance bottleneck.

Memory Bandwidth Frames sent from or received by the host result in XMI bus transactions that are written to or read from the host memory. Throughput varies with the memory implementation and interleaving. We assumed that the memory implementation and interleaving were selected such that no overloading of the memory occurs, thus eliminating wasted bus cycles.

Buffer Alignment and Segmentation We assumed that data for transmission and buffers for reception were hexaword (i.e., 32-byte) aligned and that frames were unsegmented.

Simulation Traffic No error frames or error transactions were simulated, since we assumed these to be negligible. No adapter manager traffic was simulated during the performance measurements, since these represent a very negligible fraction of the frames received during steady-state ring operation.

Throughput Measurements

Measurements were made to determine the throughput that the adapter can sustain for received and transmitted frames. It is important to understand how throughput is related to the load, the burstiness of frame arrivals, the percent XMI interference, and the frame size. This section presents the results of the throughput measurements as functions of these parameters.

Received Throughput as a Function of the Load The graph shown in Figure 3 is the result of several experiments conducted by varying the load for 33-byte received frames. The frame arrival rates depend on the load and the arrival rate distribution. As mentioned earlier, the model is capable of simulating traffic with different arrival patterns. Figure 3 shows that, with an exponential arrival pattern, the throughput increases at a rate proportional to the load up to a certain point, and then gradually decreases until the load is 100 Mb/s. The decrease in throughput is caused by the loss of resources due to excessive loading.

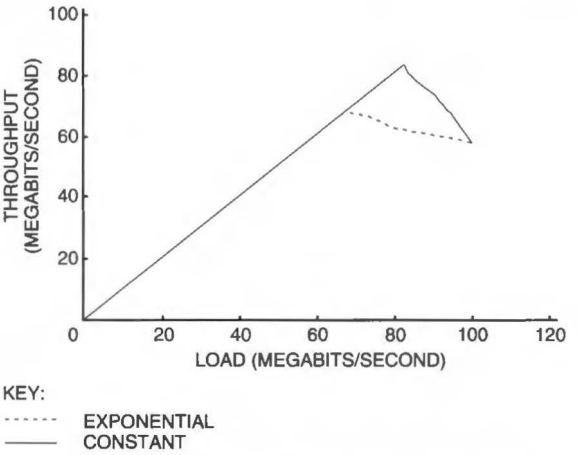


Figure 3 Receive Throughput as a Function of the Load for a 33-byte Frame

We simulated traffic with a constant arrival pattern and conducted the same experiments. These results are also shown in Figure 3. Observe that the point of maximum throughput and the rate at which the throughput decreases after reaching the maximum vary with the arrival pattern of traffic. After performing experiments on other frame sizes, we concluded that there is no fixed relationship between the maximum achievable throughput and the throughput at FDDI saturation (i.e., 100-Mb/s load). Also, there is graceful degradation in throughput after the peak.

Receive Throughput for Four- and Five-mode Workloads We measured adapter receive throughput for four- and five-mode workloads with a load of 100 Mb/s. The XMI interference workload was varied, and the results are presented in Figure 4. The adapter can receive the workload at 100 Mb/s, if the XMI interference workload remains moderate. Figure 4 also shows that there is very little difference in performance between the four- and five-mode workloads. Large frames constitute a major part of both workloads, and larger frames can be easily supported by DEMFA at full FDDI data bandwidth.

Receive Throughput as a Function of Frame Size Figure 5 shows the throughput as a function of the frame size and the XMI interference workload, with DEMFA attached to an XMI (CPU) bus. Smaller frames

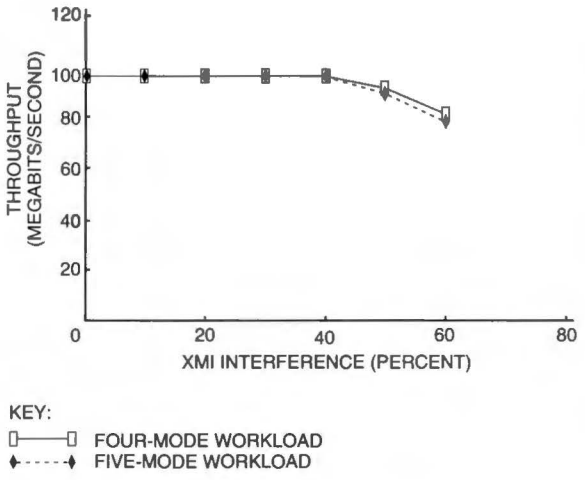


Figure 4 Receive Throughput as a Function of XMI Interference for an XMI (CPU) Bus Configuration

have a lower throughput rate than larger ones because of high control/data overhead. Since control transactions consume bandwidth, the bandwidth available for data movement is reduced. Consequently, the overall throughput rate is lower. Another reason for lower adapter throughput is the XMI utilization by traffic from other nodes on the XMI bus. This XMI interference results in less available XMI bandwidth for the adapter and hence, less throughput.

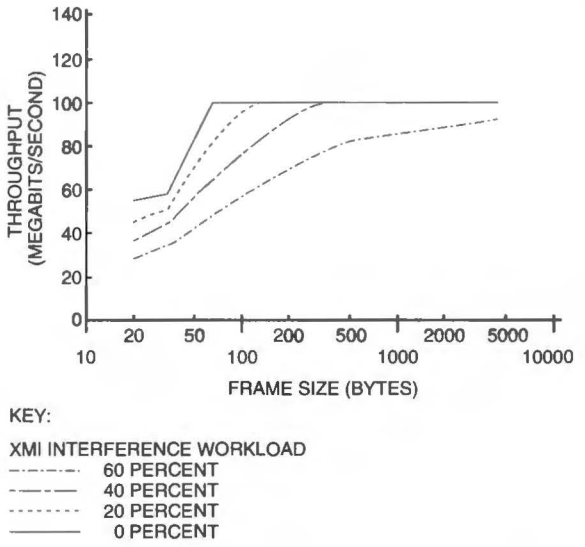


Figure 5 Receive Throughput as a Function of the Frame Size for an XMI (CPU) Bus Configuration

The adapter throughput for an XMI (I/O) bus configuration differs only slightly from that for an XMI (CPU) bus configuration. Any differences that exist are for frames smaller than 64 bytes, since the adapter experiences a per-frame latency cost because the memory is not local to the XMI bus.

Transmit Throughput for Four- and Five-mode Workloads Figure 6 illustrates the transmit throughput for a four-mode workload as a function of the XMI interference. We performed simulations to obtain throughput data for the DEMFA when attached to an XMI (CPU) bus or to an XMI (I/O) bus. Throughput for the XMI (CPU) bus configuration is 100 Mb/s and is insensitive to low, XMI interference loads. Whereas, XMI (I/O) bus configuration measurements are negatively affected by all levels of XMI interference traffic. The higher read latency that is inherent to an XMI (I/O) bus configuration degrades further with increasing interference traffic. In addition the degradation appears to be linear. The throughputs observed for the five-mode workloads are very similar to the data shown in Figure 6.

Transmit Throughput as a Function of the Frame Size Figure 7 shows the throughput as a function of the frame size when the DEMFA is attached to an XMI (CPU) bus. Throughput is also presented for various XMI interference workloads. As in the case of receive throughput, transmit throughput degrades as the frame size decreases and the XMI interference load increases. This degradation is

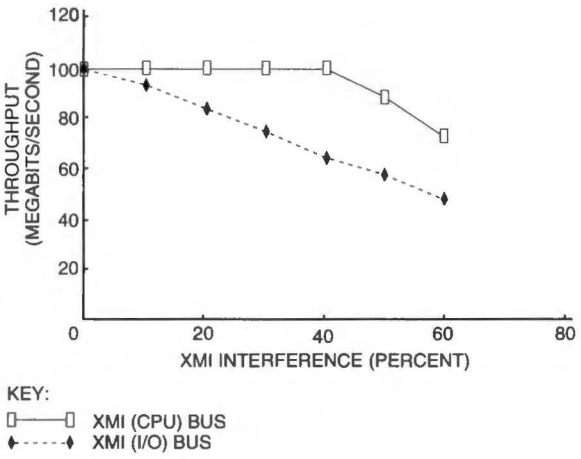


Figure 6 Transmit Throughput as a Function of XMI Interference for a Four-mode Workload

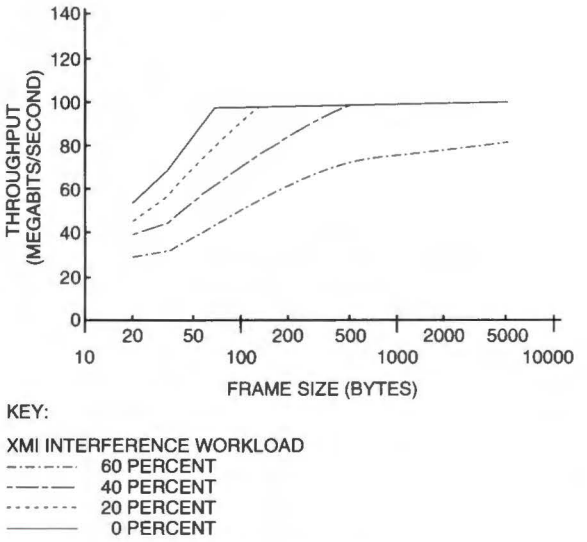


Figure 7 Transmit Throughput as a Function of the Frame Size for an XMI (CPU) Bus Configuration

again attributed to high control/data overhead and lower XMI bandwidth availability.

Figure 8 shows adapter transmit throughput as a function of the frame size for an XMI (I/O) bus configuration. The transmit throughput is less when the DEMFA is used with an XMI (I/O) bus rather than with an XMI (CPU) bus, due to the larger amount of read access time resulting from the XMI (I/O) bus configuration. The transmit operation consists mainly of read transactions and hence, this latency is crucial to transmit performance.

Latency Measurements

Latency, as it relates to the DEMFA, is explained in the Definition of Metrics section. We measured the latency for receive and transmit frames. Frame latency consists of two components: the active component, which contributes to the time when the frame or a portion thereof is being processed at a service center (also called the service time); and the passive component, which is the time when the frame or a portion thereof waits for access to the service center. All latency data presented in this section represents averages across a large number of samples. When measuring the latency of a frame, we applied the maximum load that can be sustained continuously for that frame size and type.

Receive Latency as a Function of the Frame Size

Figure 9 represents the receive latency data as a function of the frame size for an XMI (CPU) bus configuration. Latency is also presented for various XMI interference levels. We present performance data for only one XMI configuration because there is little variation between the results for the XMI (CPU) bus and XMI (I/O) bus configurations. Both frame

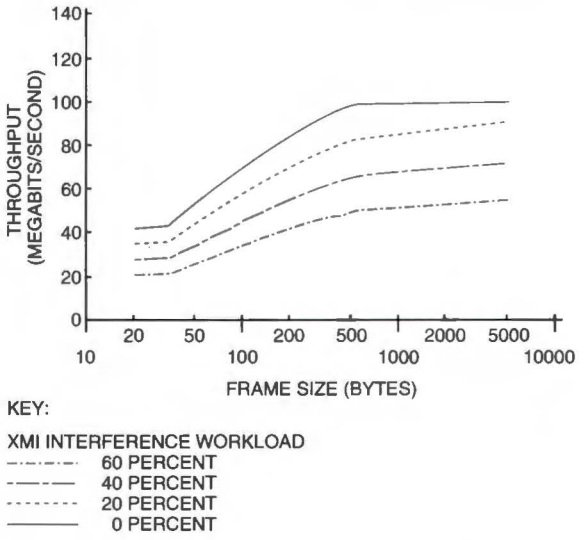


Figure 8 Transmit Throughput as a Function of the Frame Size for the XMI (I/O) Bus Configuration

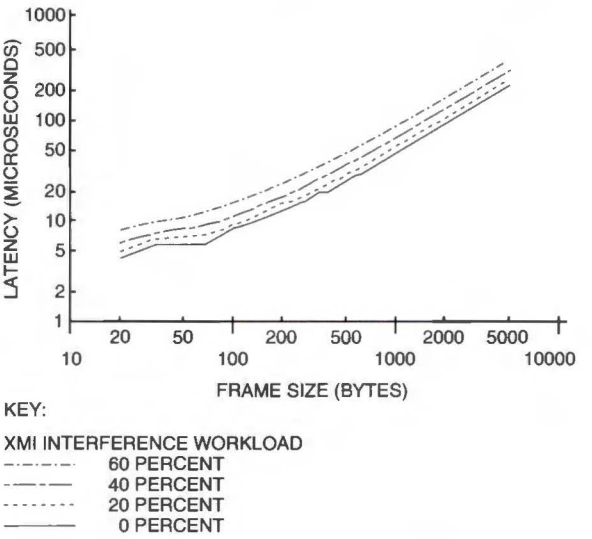


Figure 9 Receive Latency as a Function of the Frame Size for an XMI (CPU) Bus Configuration

size and latency are plotted using logarithmic scales. The data illustrates that XMI latency increases linearly with increased XMI interference.

Transmit Latency as a Function of the Frame Size

Figure 10 presents transmit latency results for an XMI (CPU) bus configuration and Figure 11 presents the results for an XMI (I/O) bus configuration. The latency was measured as a function of the frame size for various XMI interference workloads. Transmit latency is more sensitive to the system type and to the XMI interference workload because most XMI transactions that constitute transmit traffic are read operations. There is a distinctly higher latency cost associated with these transactions in the XMI (I/O) bus configuration as compared to the XMI (CPU) bus configuration. As in the case of receive latency, the transmit latency degrades with XMI interference.

Performance Measurements with the Prototype DEMFA

The intent of performing measurements with the prototype DEMFA was twofold. First, we wanted to confirm the performance predictions arrived at through simulation. And second, we wanted to measure some features that we did not implement in the model, either because they were not quantifiable or because they were too complex to model.

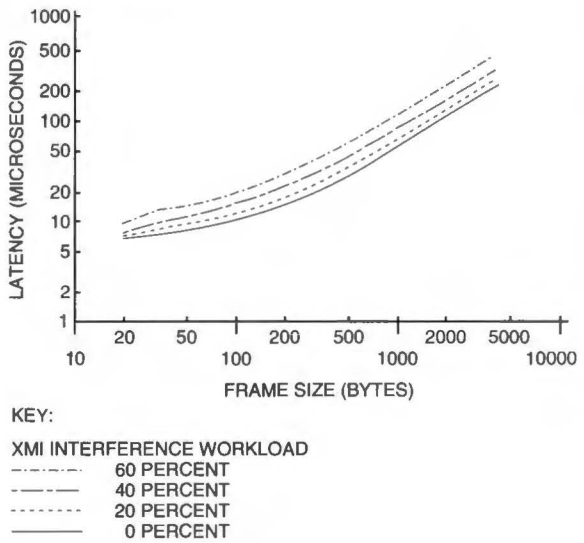


Figure 10 Transmit Latency as a Function of the Frame Size for an XMI (CPU) Bus Configuration

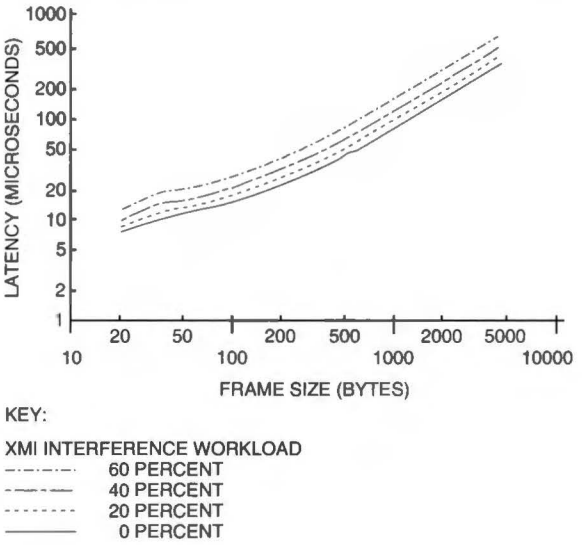


Figure 11 Transmit Latency as a Function of the Frame Size for an XMI (I/O) Bus Configuration

Again, we present only hardware performance measurements; system performance with the DEMFA is beyond the scope of this paper.

Measurement Setups

The experimental configuration required to perform the measurements on the prototype DEMFA is shown in Figure 12. This configuration consists of a VAX 6000 processor connected to a DECconcentrator 500. The VAX 6000 system has an XMI backplane. The DEMFA occupies one of the

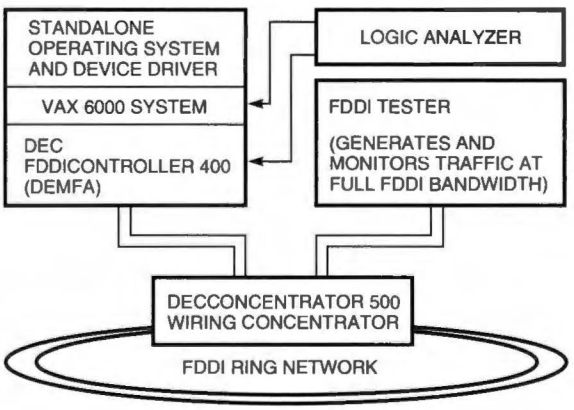


Figure 12 Laboratory Setup for DEMFA Performance Measurements

slots in the XMI backplane and is part of the XMI (CPU) bus configuration in this system. An FDDI tester is also attached to the DECconcentrator 500 and acts as a source of frames. The FDDI tester is a useful tool for testing the DEMFA product; the tester is capable of transmitting traffic at 100 Mb/s and can generate frames of various sizes and types with different destination addresses. A standalone software driver and operating system runs on the VAX 6000 system and is used for DEMFA hardware performance tests. A logic analyzer is used to measure elapsed time and count events.

Throughput Measurements

The device driver measures receive and transmit throughput and is designed to perform minimal processing for each frame.

Receive Throughput Measurements We measured the receive throughput by sending a continuous stream of frames at 100 Mb/s from the FDDI tester to the DEMFA. We varied the frame size for the tests and ran each test for a length of time sufficient to verify data convergence.

We compared the prototype measurements with the modeled results for receive throughput as a function of the frame size for an XMI (CPU) bus configuration. This validation of the receive throughput results is shown in Figure 13. The hardware measurements demonstrate that the adapter can receive frame sizes above 69 bytes at 100 Mb/s. Throughput degrades for smaller frame sizes. These measurements closely validate the modeled results. The

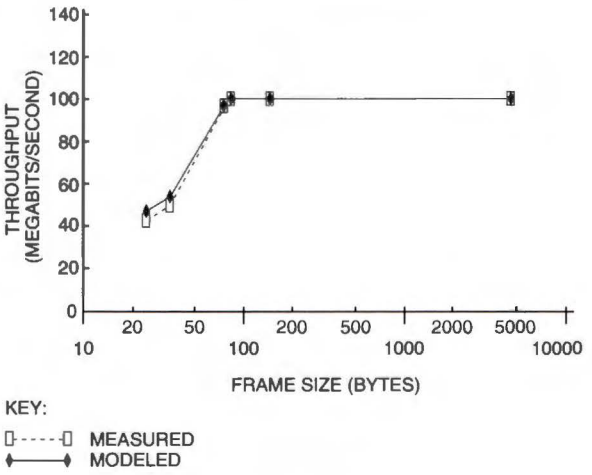


Figure 13 Validation of Receive Throughput Results

throughput for the performance model demonstrates that the DEMFA can continuously receive frames greater than 65 bytes at 100 Mb/s. There is a slight difference between the measured and modeled results at the lower frame sizes because residual XMI interference traffic exists in the measured system. This experimental error is unavoidable, but the difference is a small percentage of the total throughput and is therefore acceptable.

Transmit Throughput Measurements To measure the transmit throughput, we forwarded frames from the driver to the FDDI ring at the maximum possible rate. The throughput was calculated from the number of frames that could be sent in a unit of time. The adapter can transmit frames larger than 51 bytes at 100 Mb/s. Transmit throughputs measured in the laboratory validate the modeled results as closely as the receive throughput validation results shown in Figure 13. The modeled throughput results were lower than the measured results because we used a conservative approach to modeling the memory latency.

Multisegmented and Misaligned Frames Segmentation and alignment of transmit frame buffers in host memory is variable. Typically, frames consist of two segments, the first containing the frame header information and the second containing the data. Since the DEMFA must access control and data separately, segmentation makes this process less efficient, from a hardware perspective, than if the data and control information exist in the same buffer. Also, buffers may be aligned to start on different byte boundaries. Since the DEMFA transactions begin on hexaword (i.e., 32-byte) boundaries, hexaword alignment of frame data in the host buffers is the most efficient arrangement from the adapter's perspective. We measured throughput with unsegmented and two-segmented frames, and with frames aligned on longword, quadword, and hexaword byte boundaries. Segmentation and alignment variations cause negligible throughput degradation for frames 64 bytes or larger.

Latency Measurements

We used the logic analyzer to measure the frame latency. The logic analyzer responds to signals that indicate the starting and ending times for processing a frame. The difference between these two times is the frame latency. The events were chosen such that the measurements conformed to the definition

of latency as described in the Definition of Metrics section.

Note that the traffic pattern used to measure latency in this section differs from the workload illustrated in the section Performance Results from Simulation. Here, a single frame was received or transmitted, and we measured latency due to that frame only. Whereas previously, we used the simulation model to measure latency as an average across a large number of frames representing a load equal to the maximum sustainable adapter throughput. The workloads differ because of the practical difficulty to perform latency measurements on a large number of frames.

Receive Latency The receive frame latency predictions from the performance model and adapter service time measurements taken from the prototype hardware are shown in Figure 14. These latency measurements validate the model predictions in a way similar to that for the throughput measurements.

Transmit Latency We also compared transmit latency measurements to predictions from the performance model and found these measurements to approximate the modeled results. But actual latency measurements were slightly lower than the modeled results, again due to a conservative modeled latency.

Conclusions

The performance model was intended to track the performance of the prototype hardware to an accuracy of ±10.0 percent. The comparisons between modeled and measured results demonstrate that the model actually surpasses our goal. The measured performance for the XMI (I/O) bus configuration using a VAX 9000 system validated the modeled results as closely as did the corresponding results for the XMI (CPU) bus configuration. Disparity, if any, between the modeled and the measured results basically stem from unavoidable measurement errors for receive frames and pessimistic memory latency assumptions for transmit frames.

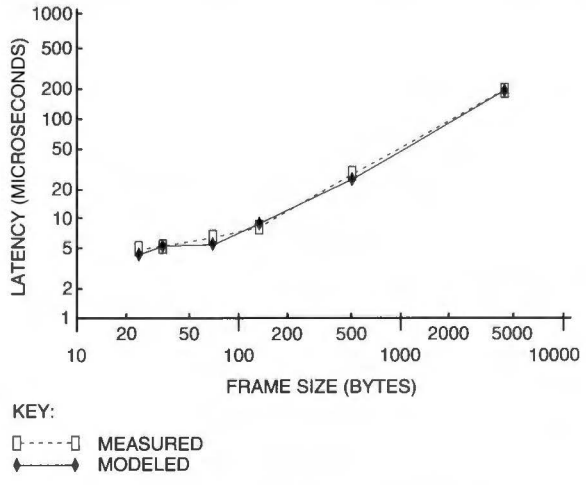


Figure 14 Validation of Receive Frame Latency Results

Throughput due to the four- and five-mode workloads is nearly the same. The average frame size for these distributions is 496 bytes and 487 bytes, respectively. Thus, throughput is a function of the frame size and independent of the number of modes that exist in the workload. Also, this data leads to the conclusion that the DEMFA may never pose as a performance bottleneck in a real network environment.

For the simulation, we chose an XMI workload with an extremely high burst rate. Actual XMI systems may result in better throughput than that presented in this paper. The resources required to create XMI workload variations are not easily accessible, so we did not perform measurements on the prototype adapter under different workload conditions. But since other measurements validated the model predictions so closely, measuring performance with varied XMI workloads proved unnecessary.

Validation of the results that we predicted through simulation increased our confidence in various design mechanisms that were verified using the performance model as a test platform. When designing new I/O architecture or memory implementations, our performance model allows changes to be made easily in order to determine the impact of such changes on performance. The modeling strategy proved very effective and helped to deliver a high-quality product with better performance than what was intended initially.

Acknowledgments

I wish to acknowledge all members of the DEMFA development group for their help in modeling the adapter. Their openness to examine new designs to enhance performance resulted in this high-speed adapter. I am also grateful to the group for assisting with the performance measurements. Finally, I wish to extend special thanks to Howard Hayakawa, Gerard Koeckhoven, Satish Rege, Andy Russo, and Dick Stockdale.

References

1. R. Gillett, "Interfacing a VAX Microprocessor to a High-speed Multiprocessing Bus," *Digital Technical Journal,* no. 7 (August 1988): 28–46.

2. W. Hawe, R. Graham, and P. Hayden, "Fiber Distributed Data Interface Overview," *Digital Technical Journal,* vol. 3, no. 2 (Spring 1991): 10–18.

3. B. Allison, "An Overview of the VAX 6200 Family of Systems," *Digital Technical Journal,* no. 7 (August 1988): 19–27.

4. D. Fite, Jr., T. Fossum, and D. Manley, "Design Strategy for the VAX 9000 System," *Digital Technical Journal,* vol. 2, no. 4 (Fall 1990): 13–24.

5. F. Ross, "FDDI—A Tutorial," *IEEE Communications Magazine,* vol. 24, no. 5 (May 1986): 10–17.

6. S. Joshi, "High-Performance Networks: A Focus on the Fiber Distributed Data Interface (FDDI) Standard," *IEEE MICRO* (June 1986): 8–14.

7. R. Stockdale and J. Weiss, "Design of the DEC LANcontroller 400 Adapter," *Digital Technical Journal,* vol. 3, no. 3 (Summer 1991, this issue): 36–47.

8. S. Rege, "The Architecture and Implementation of a High-performance FDDI Adapter," *Digital Technical Journal,* vol. 3, no. 3 (Summer 1991, this issue): 48–63.

9. *Programmer's Reference Manual for SIMULA for VAX under VMS Operating System* (North Berwick, Scotland: EASE Ltd., 1991).

10. G. Birtwistle, O. Dahl, B. Myhrhaug, and K. Nygaard, *SIMULA BEGIN* (Kent, England: Chartwell-Bratt Ltd., 1980).

11. L. Kleinrock, *Queueing Systems,* vols. 1 and 2 (New York: John Wiley and Sons, 1976).

12. D. Chiu and R. Sudama, "A Case Study of DECnet Applications and Protocol Performance," *Proceedings of the ACM SIGMETRICS Conference* (May 1988).

13. H. Yang, B. Spinney, and S. Towning, "FDDI Data Link Development," *Digital Technical Journal,* vol. 3 no. 2 (Spring 1991): 31–41.

Raj Jain

Performance Analysis of FDDI

The performance of an FDDI LAN depends upon configuration and workload parameters such as the extent of the ring, the number of stations on the ring, the number of stations that are waiting to transmit, and the frame size. In addition, one key parameter that network managers can control to improve performance is the target token rotation time (TTRT). Analytical modeling and simulation methods were used to investigate the effect of the TTRT on various performance metrics for different ring configurations. This analysis demonstrated that setting the TTRT at 8 milliseconds provides good performance over a wide range of configurations and workloads.

Fiber distributed data interface (FDDI) is a 100-megabit-per-second (Mb/s) local area network (LAN) defined by the American National Standards Institute (ANSI).[1,2] This standard allows as many as 500 stations to communicate by means of fiber-optic cables using a timed-token access protocol. Normal data traffic and time-constrained traffic, such as voice, video, and real-time applications, are supported. All major computer and communications vendors and integrated circuit manufacturers offer products that comply with this standard.

Unlike the token access protocol defined by the IEEE 802.5 standard, the timed-token access protocol used by FDDI allows synchronous and asynchronous traffic simultaneously.[3] The maximum access delay, i.e., the time between successive transmission opportunities for a station, is bounded for both types of traffic. Although this delay is short for synchronous traffic, that for asynchronous traffic varies with the network configuration and load and can be as long as 165 seconds. Long maximum access delays are undesirable and can be avoided by properly setting the network parameters and configurations.

This paper begins with a description of the timed-token access method used by FDDI stations and then proceeds to discuss how various parameters affect the performance of these systems. The target token rotation time (TTRT) is one of the key parameters. We investigated the effect of the TTRT on FDDI LAN performance and developed guidelines for setting the value of this parameter. The results of our investigation constitute a significant portion of this paper.

This paper is a modified version of "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," by Raj Jain, published in the *Proceedings of the SIGCOMM '90,* September 1990.

Timed-token Access Method

The token access method for network communication, as defined by the IEEE 802.5 standard, operates in the following manner. A token circulates around the ring network. A station that wants to transmit information waits for the arrival of the token. Upon receiving the token, the station can transmit for a fixed time interval called the token holding time (THT). The station releases the token either immediately after completing transmission or after the arrival of all the transmitted frames. The time interval between two successive receptions of the token by a station is called the token rotation time (TRT). Using this scheme, a station on an n-station ring may have to wait as long as n times the THT interval to receive a token. This maximum access delay may be unacceptable for some applications if the value of either n or THT is large. For example, voice traffic and real-time applications may require that this delay be limited to 10 to 20 milliseconds (ms). Consequently, using the token access method severely restricts the number of stations on a ring.

The timed-token access method, invented by Grow, solves this problem by ensuring that all stations on a ring agree to a target token rotation time (TTRT) and limit their transmissions to meet this target.[4] There are two modes of transmission:

synchronous and asynchronous. Time-constrained applications such as voice and real-time traffic use the synchronous mode. Traffic that does not have time constraints uses the asynchronous mode. A station can transmit synchronous traffic whenever it receives a token; however, the total transmission time for each opportunity is short. This time is allocated at the ring initialization. A station can transmit asynchronous traffic only if the TRT is less than the TTRT.

The basic algorithm for asynchronous traffic is as follows. All stations on a ring agree on a target token rotation time. Each station measures the time elapsed since last receiving the token, i.e., the TRT. On token arrival, a station that wants to transmit computes a token holding time using the following formula:

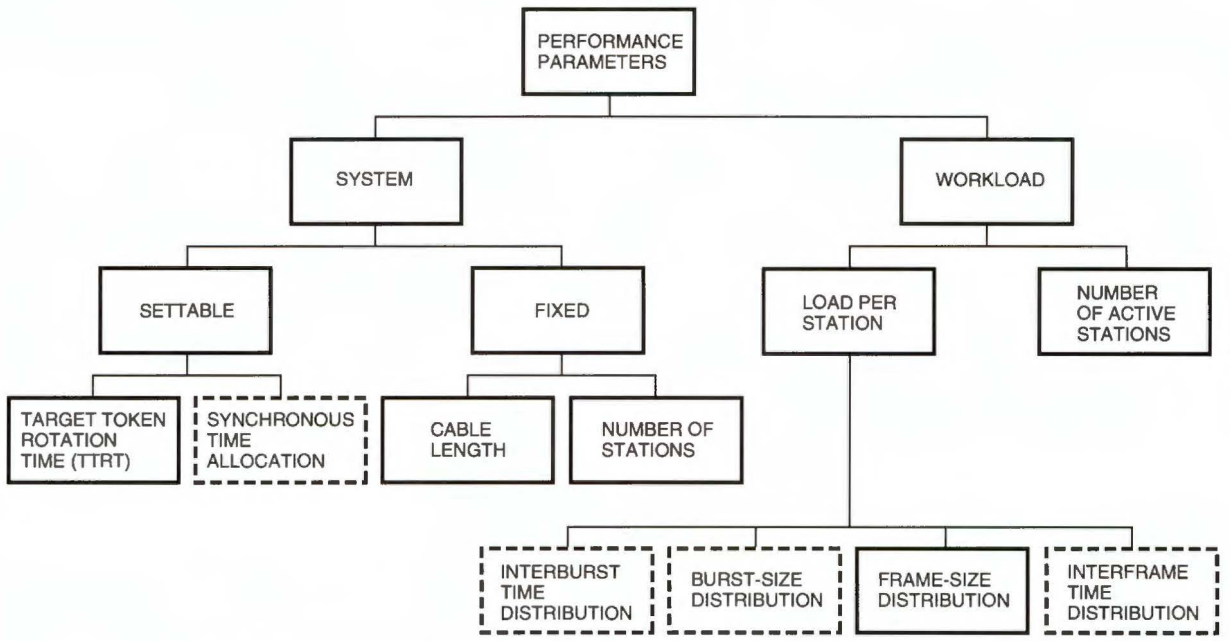
$$THT = TTRT - TRT$$

If the value of THT is positive, the station can transmit for this time interval. After completing transmission, the station releases the token. If a station does not use its entire THT, other stations on the ring can use the remaining time through successive applications of the algorithm.

Note that even though the stations attempt to keep the TRT below the target, they do not always achieve this goal. The TRT can exceed the target by as much as the sum of all synchronous-transmission time allocations; however, these allocations are limited so that their sum is less than the TTRT. As a result, the TRT is always less than twice the TTRT.

Performance Parameters

The performance of any system depends upon both system parameters and workload parameters as shown in Figure 1. There are two kinds of system parameters: fixed and user-settable. Fixed parameters cannot be controlled by the network manager and vary from one ring to another. Cable length and the number of stations on a ring are examples of fixed parameters. It is important to study network performance with respect to these parameters; if performance is sensitive to them, each set of fixed parameters may require a different guideline. System parameters that can be set by the network manager or the individual station manager include various timer values. Most of these timers influence the reliability of the ring and the time it takes to detect a malfunction. The key settable parameters



Note that the parameters shown in the dashed boxes were not considered in this study.

Figure 1 Performance Parameters

that affect system performance are the TTRT and the synchronous time allocations.

In this paper, the performance was studied under asynchronous traffic conditions only. The presence of synchronous traffic will further restrict the choice of TTRT, as discussed later in the section Guidelines for Setting the Target Token Rotation Time.

The workload also has a significant impact on performance. A guideline or recommendation may be suitable for one workload but not for another. The key workload parameters are the number of active stations and the load per station. By active we mean stations on a ring that are either transmitting or waiting to transmit. A ring may contain a large number of stations, but generally only a few are active at any given time. Active stations include the currently transmitting station, if any, and stations that have frames to transmit and are waiting for the access right, i.e., for a usable token to arrive. The load per station varies with the number of stations, the interval between bursts, the number of frames per burst, and the frame size.

Performance Metrics

The quality of service a system provides is measured by its productivity and responsiveness.[5] For an FDDI LAN, productivity is measured by its throughput, and responsiveness is measured by the response time and maximum access delay.

The productivity metric of concern to the network manager is the total throughput measured in megabits per second. Over any reasonable time interval and for most loads, the throughput is equal to the load. For example, if the load on a ring is 40 Mb/s, then the throughput is also 40 Mb/s. But this does not hold if the load is high. For example, if there are three stations on a ring, each with a 100-Mb/s load, the total arrival rate is 300 Mb/s and the throughput is considerably less—close to 100 Mb/s. Thus, the key productivity metric is not the throughput under low load but the maximum obtainable throughput under high load. This latter quantity is also known as the usable bandwidth of the network. And the ratio of the usable bandwidth to the nominal bandwidth (e.g., 100 Mb/s for an FDDI LAN) is called the efficiency of the network. For instance, if we consider a set of configuration and workload parameters with a usable FDDI bandwidth of at most 90 Mb/s, the efficiency is 90 percent.

The response time is the time interval between the arrival of a frame and the completion of its transmission, including queuing delay, i.e., from the first bit in to the last bit out. This metric is meaningful only if a ring is not saturated, because at loads near or above capacity the response time approaches infinity. With such loads, the maximum access delay for a station, i.e., the time interval between wanting to transmit and receiving a token, has more significance.

Another metric that is of interest for a shared resource such as FDDI is the fairness with which the resource is allocated. Fairness is particularly important under a heavy load. Given such a load, the asynchronous bandwidth is allocated equally to all active stations. However, the FDDI protocols are fair only if the priority levels are not implemented. In the case of multiple priority implementation, it is possible for two stations with the same priority and the same load to have different throughput depending upon their location.[6] Low-priority stations that are close to high-priority stations may get better service than those farther downstream. We assumed a single priority implementation to keep the analysis simple. Since such implementations have no fairness problem, this metric will be discussed no further in this paper.

We used two methods to analyze performance: analytical modeling and simulation. We first present the analytical model used to compute the efficiency and maximum access delay of a network under a heavy load. Then we discuss the simulation model workload used to analyze the response time at loads below the usable bandwidth.

A Simple Analytical Model

The maximum access delay and efficiency are meaningful only under heavy load. Therefore, we assume that there are n active stations, each generating enough frames to saturate the FDDI network.

Basic Equations

For an FDDI network with a ring latency D (i.e., the time it takes a bit to travel around the ring) and a TTRT value of T, the efficiency and maximum access delay are computed using the following formulas:

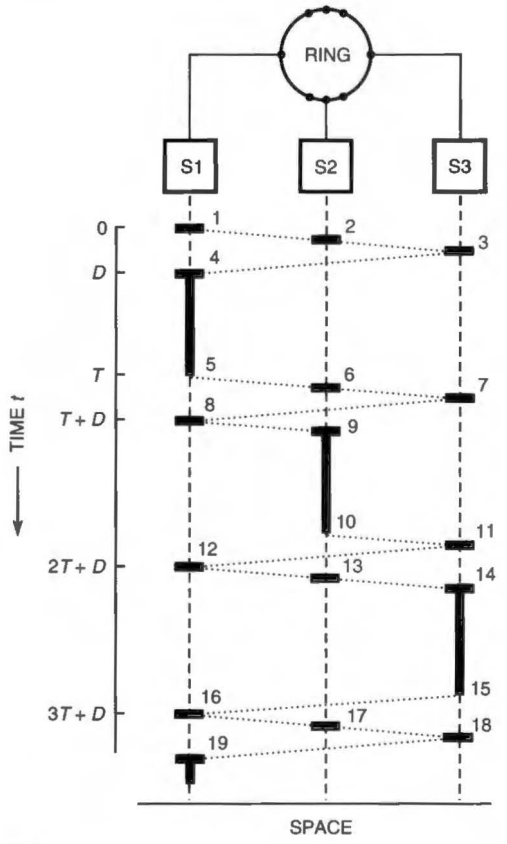
$$\text{Efficiency} = \frac{n\,(T - D)}{n\,T + D} \qquad (1)$$

$$\text{Maximum access delay} = (n - 1)\,T + 2D \qquad (2)$$

Equations (1) and (2) constitute the analytical model. Their derivation is simple and is presented in the next section. Those readers who are not interested in the details can proceed to the section Application of the Model.

Derivation

First consider a ring with three active stations, as shown in Figure 2. (Later, we will consider the general case of n active stations.) The figure shows the space-time diagram of various events on the ring. The space is shown horizontally, and the time t is shown vertically. The token reception is denoted by a thick horizontal line segment. The interval of time during which transmission of frames takes place is indicated by a thick vertical line segment.



Note that the numbers in this figure refer to event numbers discussed in the text.

Figure 2 Space-time Diagram of Events with Three Active Stations on an FDDI Network

Assume that all stations are idle until $t = D$, when the three active stations suddenly get a large (infinite) burst of frames to transmit. The sequence of events shown in Figure 2 is as follows:

1. $t = 0$. Station S1 receives the token and resets its own token rotation timer to zero. Since the station has no frames to transmit, the token proceeds to the next station S2.

2. $t = t_{12}$. Station S2 receives the token and resets its token rotation timer to zero. t_{12} is equal to the signal propagation delay from S1 to S2.

3. $t = t_{13}$. Station S3 receives the token and resets its token rotation timer to zero. t_{13} is equal to the signal propagation delay from S1 to S3.

4. $t = D$. Station S1 receives the token. Since S1 now has an infinite supply of frames to transmit, it captures the token and determines that the TRT is D. Thus, the time interval during which S1 can hold the token, the difference between TTRT and TRT, is $T - D$.

5. $t = T$. The THT at station S1 expires. S1 releases the token.

6. $t = T + t_{12}$. Station S2 receives the token. S2 last received the token at $t = t_{12}$; thus, the value of TRT is T. S2 cannot use the token at this time and releases it.

7. $t = T + t_{13}$. Station S3 receives the token. S3 last received the token at $t = t_{13}$; thus, its TRT is also T. S3 cannot use the token at this time and releases it.

8. $t = T + D$. Station S1 receives the token. S1 last received the token at $t = D$; its TRT is also T. (Note that the TRT is measured from the instant the token arrives at a station's receiver, i.e., event 4 for station S1, and not from the time it leaves a station's transmitter, i.e., event 5.) S1 cannot use the token and releases it.

9. $t = T + D + t_{12}$. Station S2 receives the token. Since TRT is only D, it sets the THT to the remaining time, i.e., $T - D$. S2 transmits for that interval and releases the token at $t = T + D + t_{12} + (T - D)$.

10. $t = 2T + t_{12}$. The THT at station S2 expires. S2 releases the token.

11. $t = 2T + t_{13}$. Station S3 receives the token. Since TRT is T, S3 releases the token.

12. $t = 2T + D$. Station S1 receives the token. Since TRT is T, S1 releases the token.

13. $t = 2T + D + t_{12}$. Station S2 receives the token. Since TRT is T, S2 releases the token.

14. $t = 2T + D + t_{13}$. Station S3 receives the token. Since TRT is only D, it transmits for the time interval $T - D$ and releases the token at $t = 2T + D + t_{13} + (T - D)$.

15. $t = 3T + t_{13}$. The THT at station S3 expires. S3 releases the token.

16. $t = 3T + D$. Station S1 receives the token, and the sequence of events begins to repeat. The token passes through stations S1, S2, and S3, all of which find it unusable.

.
.
.

19. $t = 3T + 2D$. The cycle continues with S1 capturing the token as in event 4.

The above discussion illustrates that the system goes through a cycle of events and that the cycle time is $3T + D$. During every cycle, each of the three stations transmits for a time interval equal to $T - D$; the total transmission time is $3(T - D)$. The number of bits transmitted during this time is $3(T - D) \times 10^8$, and the throughput equals $3(T - D) \times 10^8 / (3T + D)$ bits per second. The efficiency, i.e., the ratio of the throughput to the FDDI bandwidth of 100 Mb/s, is $3(T - D) / (3T + D)$.

During the cycle, each station waits for a time interval of $2T + 2D$ after releasing the token for another opportunity to transmit. This interval is the maximum access delay. For lower loads, the access delay is shorter.

Thus, for a ring with three active stations,

$$\text{Efficiency} = \frac{3(T - D)}{3T + D}$$

$$\text{Maximum access delay} = (3 - 1)T + 2D = 2T + 2D$$

To generalize the above analysis for n active stations, substitute n for 3. Equations (1) and (2) are the results; the derivation is complete.

Application of the Model

Equations (1) and (2) can be used to compute the maximum access delay and the efficiency for any FDDI ring configuration. For example, consider a ring with 16 stations and a total fiber length of 20 kilometers (km). (Using a two-fiber cable, this corresponds to a cable length of 10 km.) Light waves travel along the fiber at a speed of 5.085 microseconds per kilometer (μs/km). The station delay between receiving and transmitting a bit is approximately 1 μs per station. The ring latency can be computed as follows:

$$\begin{aligned}\text{Ring latency } D &= (20 \text{ km}) \times (5.085\ \mu\text{s/km}) \\ &\quad + (16 \text{ stations}) \times (1\ \mu\text{s/station}) \\ &= 0.12 \text{ milliseconds (ms)}\end{aligned}$$

Assuming a TTRT of 5 ms and all 16 stations active,

$$\text{Efficiency} = \frac{16(5 - 0.12)}{16 \times 5 + 0.12} = 97.5 \text{ percent}$$

$$\text{Maximum access delay} = (16 - 1) \times 5 + 2 \times 0.12 = 75.24 \text{ ms}$$

Thus, on this ring, the maximum possible throughput is 97.5 Mb/s. If the load is greater than this for any substantial length of time, the queues will build up, the response time will increase, and the stations may start to lose frames due to insufficient buffers. The maximum access delay is 75.24 ms; thus, asynchronous stations may have to wait as long as 75.24 ms to receive a usable token.

The key advantage of this model is its simplicity, which allows us to see immediately the effect of various parameters on network performance. With only one active station, which is usually the case, equation (1) becomes

$$\text{Efficiency } (n = 1) = \frac{T - D}{T + D}$$

As the number of active stations increases, the efficiency increases. With a very large number of stations,

$$\text{Maximum efficiency } (n = \infty) = 1 - \frac{D}{T}$$

This efficiency formula is easy to remember and permits "back-of-the-envelope" calculations of FDDI LAN performance. This special case of $n = \infty$ has already been studied.[7]

Similarly, we can use equation (2) to calculate the maximum access delay with one active station as follows:

$$\text{Maximum access delay } (n = 1) = 2D$$

That is, a single active station may have to wait as long as twice the ring latency between successive transmissions because every alternate token that it receives would be unusable. For $n = \infty$, the maximum access delay approaches infinity.

Simulation Workload

One way to measure the responsiveness of a system is to use simulation to analyze the response time. This metric depends upon the frame arrival pattern of the workload and is discussed further in the Response Time section. The workload we used in our simulations was based on an actual measurement of traffic at a customer site. The chief application at this site was the warehouse and inventory control (WIC). Hence, we named the workload WIC.

Previous network measurements show that when a station wants to transmit, it generally transmits not one frame, but a burst of frames. The WIC workload has this trait as well. Therefore, we used a bursty Poisson arrival pattern in our simulation model with an interburst time of 1 ms and five frames per burst.

We limited the frames to two sizes: 65 percent of the frames were small (100 bytes), and 35 percent were large (512 bytes). This workload constitutes a total load per station of 1.22 Mb/s. Forty stations, each executing this load, would utilize 50 percent of the FDDI bandwidth. Higher load levels can be obtained either by reducing the interburst time or increasing the number of stations on the ring.

Guidelines for Setting the Target Token Rotation Time

This section presents the rules specified by the ANSI FDDI media access control standard for setting the value of the TTRT. We also discuss efficiency, maximum access delay, and response time considerations, as well as reasons to limit the value of TTRT.

ANSI FDDI Standard

According to the ANSI FDDI standard, the following rules must be observed when setting the TTRT:

1. Since the TRT can be as long as twice the TTRT, a synchronous station may have to wait a time interval of up to $2T$ before receiving the token. Therefore, synchronous stations should request a TTRT value of one-half the required service interval. For example, a voice station that wants to receive a token every 20 ms or less should request a TTRT of 10 ms.

2. The TTRT must allow transmission of at least one maximum-size frame in addition to the synchronous time allocation, if any. That is,

   $$\text{TTRT} > \text{ring latency} + \text{token time} + \text{maximum frame time} + \text{synchronous time allocation}$$

   The maximum-size frame on FDDI is 4500 bytes plus preamble and takes approximately 0.361 ms to transmit. The maximum ring latency is 1.773 ms. The token time (11 bytes including 8 bytes of preamble) is 0.00088 ms. This rule, therefore, requires that the TTRT be set at a value greater than or equal to 2.13 ms plus the synchronous time allocation. Violating this rule, for example, by overallocating the synchronous bandwidth, results in unfairness and starvation, i.e., some stations are unable to transmit.

3. A station must request a TTRT greater than or equal to the station parameter T_min. The default maximum value of T_min is 4 ms. Generally, most stations do not request a TTRT less than 4 ms.

4. A station must request a TTRT less than or equal to the station parameter T_max. The default minimum value of T_max is 165 ms. Assuming that there is at least one station with T_max equal to 165 ms, the TTRT on a ring cannot exceed this value. (In practice, many stations will use a value of $2^{22} \times 40$ ns = 167.77216 ms, which can be conveniently derived from the symbol clock using a 22-bit counter.)

Efficiency and Maximum Access Delay Considerations

In addition to the rules specified by the standard, the TTRT values should be chosen to allow high-performance operation of a ring. This section discusses these performance considerations.

Figure 3 is a plot of efficiency as a function of the TTRT. Three configurations called "Typical," "Big," and "Largest" are shown.

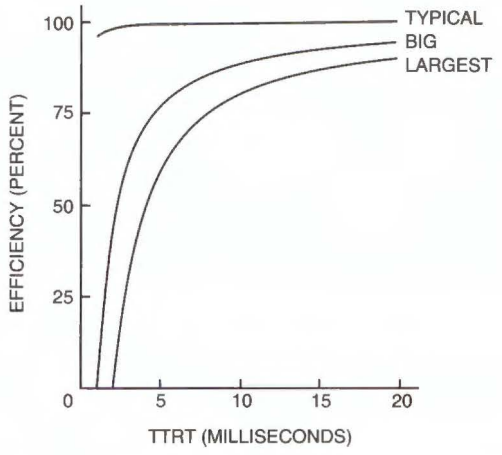


Figure 3 Efficiency as a Function of the TTRT

The Typical configuration consists of 20 single attachment stations (SASs) on a 4-km fiber ring. The numbers used are based on an intuitive feeling of what a typical ring would look like and not based on any survey of actual installations. Twenty offices located on a 50 m by 50 m floor require a 2-km cable or a 4-km fiber.

The Big configuration consists of 100 SASs on a 200-km fiber. This configuration represents a reasonably large ring with acceptable reliability. Configuring a single ring with considerably more than this number of stations increases the probability of bit errors.[8]

The Largest configuration consists of 500 dual attachment stations (DASs) and a ring that has wrapped. A DAS can have one or two media access controllers (MACs). In this configuration, each DAS has two MACs. Thus, the LAN consists of 1000 MACs in a single logical ring. This is the largest number of MACs allowed on an FDDI LAN. Exceeding this number would require recomputation of all default parameters specified in the standard.

Figure 3 shows that for all three configurations, the efficiency increases as the TTRT increases. The efficiency is very low at TTRT values close to the ring latency but increases as the TTRT increases. Thus, to ensure a minimal efficiency, the minimum allowed TTRT on FDDI is 4 ms. This direct relationship between the efficiency and the TTRT may lead some to conclude that the largest possible TTRT be chosen. However, notice also that the efficiency gained by increasing the TTRT, i.e., the slope of the efficiency curve, decreases as the TTRT increases. The "knee" of the curve depends upon the ring configuration. For larger configurations, the knee occurs at larger TTRT values. Even for the Largest configuration, the knee occurs in the range of 6 to 10 ms. For the Typical configuration, the TTRT has little effect on efficiency as long as the TTRT is in the allowed range of 4 to 165 ms.

Figure 4 shows the maximum access delay as a function of the TTRT for the three configurations. To show the complete range of possibilities, we used a semilogarithmic scale on the graph. The vertical scale is logarithmic, while the horizontal scale is linear. The figure shows that increasing the TTRT brings about a corresponding increase in the maximum access delay for all three configurations.

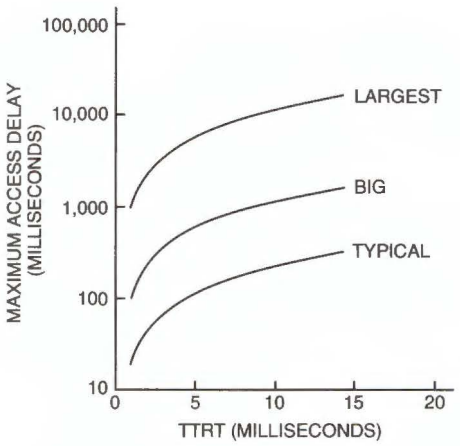


Figure 4 Maximum Access Delay as a Function of the TTRT

Table 1 presents the performance metrics for the maximum access delay and the efficiency as functions of the TTRT. As evidenced in the table, on the Largest ring, a TTRT of 165 ms causes a maximum access delay as long as 165 seconds. This means that in a worst-case situation, a station may have to wait several minutes to receive a usable token. For many applications, this delay is unacceptable; therefore, a reduced number of stations or a smaller TTRT may be preferable.

Response Time

Figure 5 shows the average response time as a function of the TTRT for a relatively large configuration, i.e., 100 stations and 10 km of fiber. The WIC workload was simulated at three load levels: 28, 58, and 90 percent. Two of the three curves are horizontal

Table 1 Maximum Access Delay and Efficiency as Functions of the TTRT

TTRT (ms)	Maximum Access Delay (seconds)			Efficiency (percent)		
	Typical 20 SAS 4 km	Big 100 SAS 200 km	Largest 500 DAS 200 km	Typical 20 SAS 4 km	Big 100 SAS 200 km	Largest 500 DAS 200 km
4	0.08	0.40	4.00	98.94	71.87	49.55
8	0.15	0.79	8.00	99.47	85.92	74.77
12	0.23	1.19	11.99	99.65	90.61	83.18
16	0.30	1.59	15.99	99.74	92.95	87.38
20	0.38	1.98	19.98	99.79	94.36	89.91
165	3.14	16.34	164.84	99.97	99.32	98.78

straight lines indicating that TTRT has no effect on the response times at these loads. The TTRT only affects the response time at a heavy load. In fact, it is only near the usable bandwidth that the TTRT has any effect on the response time.

To summarize the results presented so far, if the FDDI load is below saturation, the TTRT has little effect. At saturation, a larger value of TTRT gives a larger usable bandwidth and therefore increased efficiency. But a longer TTRT also results in longer maximum access delays. The selection of the TTRT requires a trade-off between these two requirements. To facilitate making this trade-off, the two performance metrics for the three configurations are listed in Table 1. TTRT values in the allowable range of 4 to 165 ms are shown. The data shows that a very small value of TTRT, such as 4 ms, is undesirable, because the resulting efficiency on the Largest ring is poor (50 percent). A very large value of TTRT, such as 165 ms, is also undesirable, because it results in long maximum access delays. The 8-ms value is the most desirable, since it yields 75 percent or more efficiency on all configurations and results in a maximum access delay of less than one second on Big rings. Eight milliseconds is, therefore, the recommended default TTRT.

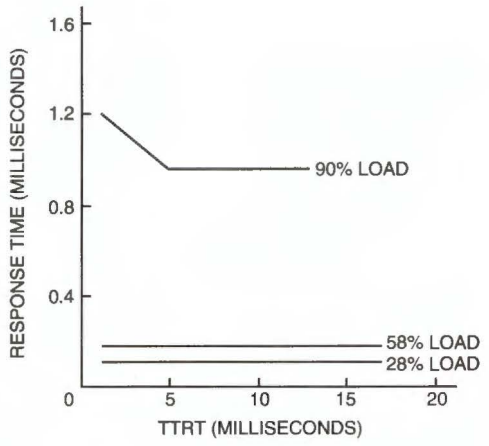


Figure 5 Response Time as a Function of TTRT

Problems with a Large TTRT

There are three additional reasons for preferring an 8-ms TTRT over a large TTRT such as 165 ms. First, a large TTRT allows a station to receive a large number of frames back-to-back. To operate in such an environment, all adapters must be designed with large receive buffers. Although memory is not considered an expensive part of a computer, its cost is significant for low-cost components such as adapters. The board space for the additional memory required by choosing a larger TTRT is considerable as are the bus holding times required for such large back-to-back transfers.

Second, a very large TTRT results in an exhaustive service discipline (i.e., all frames are transmitted in one token capture), which has several known drawbacks. For example, exhaustive service is unfair. Frames coming to higher load stations have a greater chance of finding the token during the same transmission opportunity, whereas frames arriving at low load stations may have to wait. Thus, the response time is inversely dependent upon the load, i.e., higher-load stations yield lower response times and vice versa.[9]

Third, with exhaustive service, the response time of a station is dependent upon station location

with respect to that of high-load stations. The station immediately downstream from a high-load station may obtain better throughput than the one immediately upstream.

Parameters Other Than the TTRT That Affect Performance

Many parameters other than the TTRT affect the performance of a network. This section discusses four configuration and workload parameters: the extent of the ring, the total number of stations, the number of active stations, and the frame size.

Extent of the Ring

The total length of the fiber is called the extent of the ring. The maximum allowed extent on an FDDI LAN is 200 km. Figures 6 and 7 are graphs illustrating the efficiency and maximum access delay as functions of the extent. A star-shaped ring with all stations at a fixed radius from the wiring closet is assumed. The total cable length, shown along the horizontal axis, is twice the radius times the number of stations. As is evident from the figures, rings with a larger extent have a slightly lower efficiency and a longer maximum access delay than those with smaller extents.

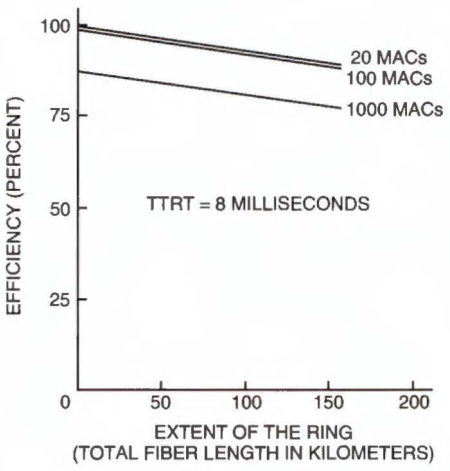


Figure 6 Efficiency as a Function of the Extent of the Ring

Note that in Figure 7, the increase in maximum access delay for each configuration is not apparent due to the semilogarithmic scale.

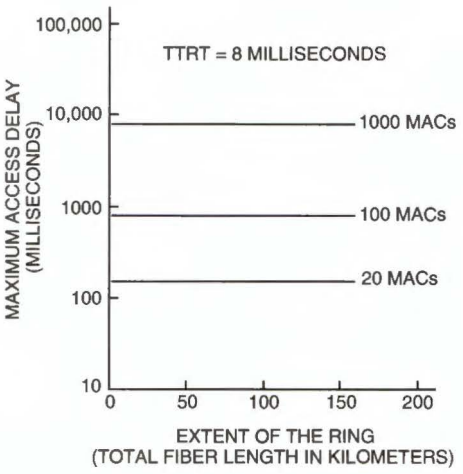


Figure 7 Maximum Access Delay as a Function of the Extent of the Ring

Total Number of Stations

The total number of stations on a ring includes active and inactive stations. In general, increasing the number of stations adds to the ring latency because of the additional fiber length and additional station delays. Thus, the number of stations affects the efficiency and maximum access delay in a way similar to that of the extent; a ring that contains a larger number of stations than another has a lower efficiency and a longer maximum access delay. In addition, a large number of stations on a ring increases the bit-error rate. Consequently, large rings are not desirable.

Number of Active Stations

As the number of active stations, i.e., MACs, increases, the total load on the ring increases. Figures 8 and 9 show the ring performance as a function of the number of active MACs on the ring. We considered a maximum-size ring with a TTRT value of 8 ms for the analysis. The figures show that increasing the number of active MACs has a slight positive effect on the efficiency, but considerably increases the maximum access delay. Therefore, it is preferable to keep a minimal number of active stations on each ring by segregating small groups on separate rings.

Frame Size

Frame size does not appear in the simple models of efficiency and maximum access delays, because

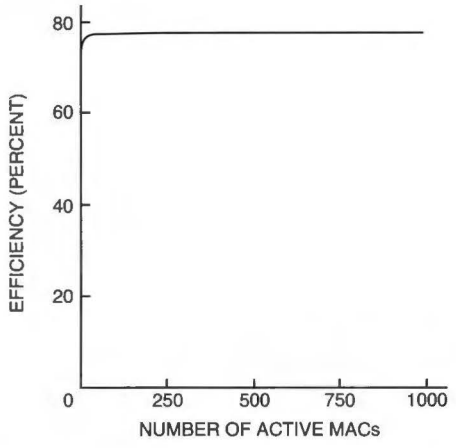


Figure 8 Efficiency as a Function of the Number of Active MACs

frame size has little impact on FDDI performance. In our analysis, we assumed that transmission stops at the instant the THT expires; however, the standard allows stations to complete the transmission of the last frame.

The extra time used by a station after THT expiry is called asynchronous overflow. Assuming all frames are of fixed size, let F denote the frame transmission time. During every transmission opportunity, an active station can transmit as many as k frames:

$$k = \left\lceil \frac{T - D}{F} \right\rceil$$

Here, $\lceil\ \rceil$ is used to denote rounding up to the next integer value. The transmission time is equal to k times F, which is slightly more than T minus D. With asynchronous overflow, the modified efficiency and maximum access delay formulas become

$$\text{Efficiency} = \frac{nkF}{n\,(kF + D) + D}$$

$$\text{Maximum access delay} = (n - 1)\,(kF + D) + 2D$$

Notice that substituting $kF = T - D$ in the above equations results in Equations (1) and (2).

Figures 10 and 11 show the efficiency and the maximum access delay as functions of the frame size. Frame size has only a slight effect on these metrics. Larger frame sizes do have the following effects:

- The probability of error is greater in a larger frame.
- Since the size of protocol headers and trailers is fixed, larger frames require less protocol overhead.
- The time to process a frame increases only slightly with the size of the frame. A larger frame size results in fewer frames and, hence, in less processing at the host.

Overall, we recommend using as large a frame size as the reliability considerations allow.

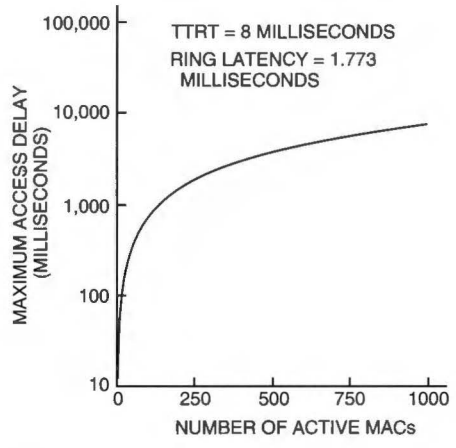


Figure 9 Maximum Access Delay as a Function of the Number of Active MACs

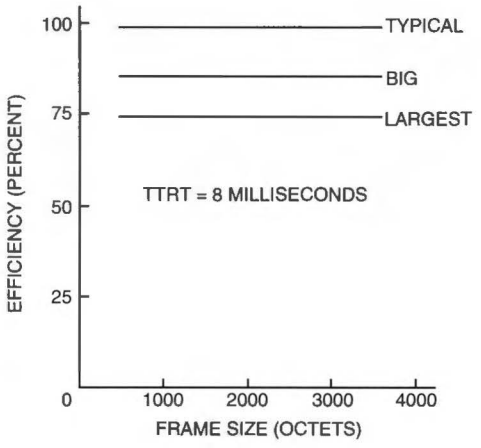


Figure 10 Efficiency as a Function of the Frame Size

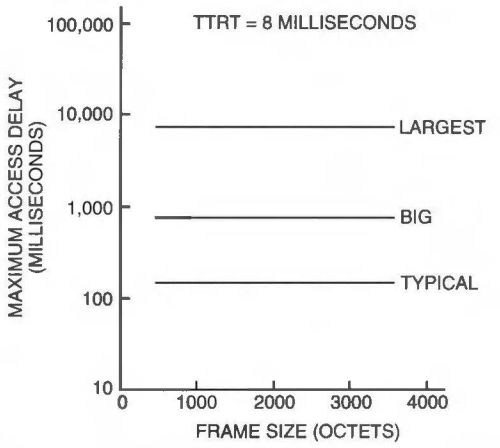


Figure 11 Maximum Access Delay as a Function of the Frame Size

Summary

Although many parameters affect the performance of an FDDI ring network, the target token rotation time (TTRT) is the key parameter that network managers can control to optimize this performance. We analyzed the effect of other parameters such as the extent of the ring (the length of the cable), the total number of stations, the number of active stations, and frame size. From our data we concluded the following:

- Rings with a large extent and those with a large number of stations are undesirable because they yield a longer maximum access delay and have only a slight positive effect on the efficiency of the ring.
- It is preferable to minimize the number of active stations on a ring to avoid increasing the maximum access delay.
- A large frame size is desirable, taking into consideration the acceptable probability of error.

The value of TTRT does not significantly affect the response time unless the load is near saturation. Under very heavy load, response time is not a suitable metric. Instead, maximum access delay, i.e., the time between wanting to transmit and being able to do so, is more meaningful.

A larger value of TTRT improves the efficiency, but it also increases the maximum access delay. A good trade-off is provided by setting TTRT at 8 ms. Since this value provides good performance for all ranges of configurations, we recommend that the default value of TTRT be set at 8 ms.

References

1. F. Ross, "An Overview of FDDI: The Fiber Distributed Data Interface," *IEEE Journal on Selected Areas in Communications,* vol. 7, no. 7 (September 1989): 1043–51.
2. *Fiber Distributed Data Interface (FDDI)—Token Ring Media Access Control (MAC),* ANSI X3.139-1987 (New York: American National Standards Institute, 1987).
3. *Token Ring Access Method and Physical Layer Specifications,* ANSI/IEEE Standard 802.5-1985, ISO/DIS 8802/5 (New York: The Institute of Electrical and Electronics Engineers, Inc., 1985).
4. R. Grow, "A Timed-token Protocol for Local Area Networks," *Proceedings of the IEEE Electro '82 Conference on Token Access Protocols,* Paper 17/3, Boston, MA (May 1982).
5. R. Jain, *The Art of Computer Systems Performance Analysis,* ISBN 0-471-50336-3 (New York: John Wiley & Sons, 1991).
6. D. Dykeman and W. Bux, "Analysis and Tuning of the FDDI Media Access Control Protocol," *IEEE Journal on Selected Areas in Communications,* vol. 6, no. 6 (July 1988): 997–1010.
7. J. Ulm, "A Timed-token Ring Local Area Network and Its Performance Characteristics," *Proceedings of the Seventh IEEE Conference on Local Computer Networks* (February 1982): 50–56.
8. R. Jain, "Error Characteristics of Fiber Distributed Data Interface (FDDI)," *IEEE Transactions on Communications,* vol. 38, no. 8 (August 1990): 1244–1252.
9. W. Bux and H. Truong, "Mean-delay Approximation for Cyclic-service Queueing Systems," *Performance Evaluation,* vol. 3 (Amsterdam: North-Holland, 1983): 187–196.

Further Readings

The Digital Technical Journal *publishes papers that explore the technological foundations of Digital's major products. Each* Journal *focuses on at least one product area and presents a compilation of papers written by the engineers who developed the product. The content for the* Journal *is selected by the Journal Advisory Board. Digital engineers who would like to contribute a paper to the* Journal *should contact the editor at RDVAX::BLAKE.*

Topics covered in previous issues of the *Digital Technical Journal* are as follows:

Fiber Distributed Data Interface
Vol. 3, No. 2, Spring 1991

Transaction Processing, Databases, and Fault-tolerant Systems
Vol. 3, No. 1, Winter 1991

VAX 9000 Series
Vol. 2, No. 4, Fall 1990

DECwindows Program
Vol. 2, No. 3, Summer 1990

VAX 6000 Model 400 System
Vol. 2, No. 2, Spring 1990

Compound Document Architecture
Vol. 2, No. 1, Winter 1990

Distributed Systems
Vol. 1, No. 9, June 1989

Storage Technology
Vol. 1, No. 8, February 1989

CVAX-based Systems
Vol. 1, No. 7, August 1988

Software Productivity Tools
Vol. 1, No. 6, February 1988

VAXcluster Systems
Vol. 1, No. 5, September 1987

VAX 8800 Family
Vol. 1, No. 4, February 1987

Networking Products
Vol. 1, No. 3, September 1986

MicroVAX II System
Vol. 1, No. 2, March 1986

VAX 8600 Processor
Vol. 1, No. 1, August 1985

Subscriptions to the *Digital Technical Journal* are available on a yearly, prepaid basis. The subscription rate is $40.00 per year (four issues). Requests should be sent to Cathy Phillips, Digital Equipment Corporation, MLO1-3/B68, 146 Main Street, Maynard, MA 01754, U.S.A. Subscriptions must be paid in U.S. dollars, and checks should be made payable to Digital Equipment Corporation.

Single copies and past issues of the *Digital Technical Journal* can be ordered from Digital Press at a cost of $16.00 per copy.

Technical Papers by Digital Authors

R. Abbott, "Scheduling I/O Requests with Deadlines: A Performance Evaluation," *IEEE Real-time Systems Symposium* (December 1990).

S. Batra, "Magnetic Superexchange in YIG & CA2:YIG," *Thirty-fifth Conference on Magnetism and Magnetic Materials* (October 1990).

A. Conn, J. Parodi, and M. Taylor, "The Place for Biometrics in a User Authentication Taxonomy," *Thirteenth National Computer Security Conference* (October 1990).

S. Das, "Suppression of Barkhausen Noise in an MR Head," *Thirty-fifth Conference on Magnetism and Magnetic Materials* (October 1990).

P. Fang, "Yield Modeling in a Custom IC Manufacturing Line," *Advanced Semiconductor Manufacturing Conference* (September 1990).

E. Freedman and Z. Cvetanovic, "Perfect Benchmarks Decomposition and Performance on VAX Multiprocessors," *IEEE Supercomputing '90* (November 1990).

L. Jaynes, "The Effect of Symbols on Warning Compliance," *Thirty-fourth Human Factors Society* (October 1990).

M. Joshi, "Making Wafers in the JIT Style," *Advanced Semiconductor Manufacturing Conference* (September 1990).

K. Mistry and B. Doyle, "Electron Traps, Interface States and Enhanced AC Hot-carrier Degradation," *IEEE Device Research* (June 1990).

Digital Press

Digital Press is the book publishing group of Digital Equipment Corporation. The Press is an international publisher of computer books and journals on new technologies and products for users, system and network managers, programmers, and other professionals. Proposals and ideas for books in these and related areas are welcomed.

The following book descriptions represent a sample of the books available from Digital Press.

VAX/VMS INTERNALS AND DATA STRUCTURES: Version 5.2

Ruth E. Goldenberg and Lawrence J. Kenah, with the assistance of Denise E. Dumas, 1991, hardbound, 1427 pages, Order No. EY-C171E-DP-EEB ($124.95)

This is a totally revised edition of the most authoritative and complete description of the VAX/VMS operating system in the industry. The book features new chapters on symmetric multiprocessing, the reorganized executive, VAX interrupts and exceptions, and the I/O subsystem, including device drivers and interrupt service routines. The addition of symmetric multiprocessing to the VAX/VMS operating system prompted major revisions to chapters concerning hardware and software interrupts, memory management, and synchronization. The authors have also taken every opportunity to clarify difficult concepts, to collect related material into single chapters, and to standardize and simplify the numerous figures contained in this reference.

VMS FILE SYSTEM INTERNALS

Kirby McCoy, 1990, softbound, 460 pages, Order No. EY-F575E-DP-EEB ($49.95)

VMS FILE SYSTEM INTERNALS, based on VMS Version 5.2, is a book for system programmers, software specialists, system managers, applications designers, and other VAX/VMS users who need to understand the interfaces to and the data structures, algorithms, and basic synchronization mechanisms of the VMS file system. This system is the part of the VAX/VMS operating system responsible for storing and managing files and information in memory and on secondary storage. The book is also intended as a case study of the VMS implementation of a file system for graduate and advanced undergraduate courses in operating systems.

VAX ARCHITECTURE REFERENCE MANUAL, Second Edition

Richard A. Brunner, Editor, 1991, softbound, 560 pages, Order No. EY-F576E-DP-EEB ($44.95)

This book describes the data types, instructions, calling standards, addressing modes, registers, exception and interrupt handling, memory management, and process structure common to all VAX computers—from the MicroVAX II to the VAX 9000. New sections describe the VAX shared-memory model supported in VAX multiprocessor computers and the recently added vector processing extensions implemented by the VAX 9000 and VAX 6000 model 400 systems. The book introduces the design goals and terminology of the VAX instruction set, including those for memory management, exception and interrupt handling, process control, and vector processing. The description of each instruction gives format, operations, condition codes, instruction-specific exceptions, opcodes, and mnemonics.

A COMPREHENSIVE GUIDE TO Rdb/VMS

Lilian Hobbs and Kenneth England, 1991, softbound, 352 pages, Order No. EY-H873E-DP-EEB ($34.95)

The Rdb/VMS relational database system was developed by Digital Equipment Corporation for VAX computers using the VMS operating system. This system is one of a number of information management products that work together to facilitate the sharing of information. The Rdb/VMS system is used, for example, in high-performance transaction processing systems. This book is based on Rdb/VMS Version 4.0, which Digital made available to customers at the end of 1990, and thus includes the latest functionality.

MIT PROJECT ATHENA: A Model for Distributed Campus Computing

George A. Champine, 1991, hardbound, 282 pages, Order No. EY-H875E-DP-EEB ($28.95)

MIT Project Athena has emerged as one of the most important models for next-generation distributed computing in an academic environment. MIT pioneered this new approach, based on the client-server model, to support a network of workstations. The project began in 1983 as a five-year project, with Digital Equipment Corporation and IBM as its two major industrial sponsors. Now a production system of networked workstations,

Project Athena is replacing time-sharing (which MIT also pioneered) as the preferred model of computing at MIT. The size and uniqueness of Project Athena has led to widespread interest in its design, implementation, and performance.

UNDERSTANDING CLOS: The Common Lisp Object System
Jo A. Lawless and Molly M. Miller, 1991, softbound, 192 pages, Order No. EY-F591E-DP-EEB ($26.95)

The Common Lisp Object System (CLOS) is an extension to Common Lisp that brings object-oriented programming (OOP) to this popular version of the Lisp language. Written for computer professionals and students, *UNDERSTANDING CLOS* quickly introduces necessary object-oriented programming concepts and provides complete syntactic descriptions of all CLOS functions adopted by the ANSI X3J13 standards committee. Also included is an 800-line sample application, as well as a bibliography, a glossary, and an index.

COMMON LISP: The Language, Second Edition
Guy L. Steele, Jr., 1990, softbound, 1029 pages, Order No. EY-C187E-DP-EEB ($38.95)

The first edition of *COMMON LISP: The Language,* which sold over 60,000 copies, became the de facto standard for the Common Lisp programming language. This second edition is approximately twice the size of the first edition. The book reflects, as closely as possible, the decisions and recommendations made by ANSI committee X3J13, bridging the gap between the first edition and the forthcoming ANSI standard. It describes many of the changes made to the Common Lisp programming language, relative to the structure of the first edition, and discusses those areas that are likely to be revised further.

To receive a copy of our latest catalog or further information on these or other publications from Digital Press, please write or call:

Digital Press
Department EEB
12 Crosby Drive
Bedford, MA 01730
(617) 276-1536

Or, you can order by calling DECdirect at 800-DIGITAL (800-344-4825).

When ordering be sure to refer to Catalog Code EEB.

Book Review

The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,
R. Jain, John Wiley & Sons, Inc., New York, 1991. 720 pages (ISBN 0-471-50336-3).

This is an edited version of a forthcoming review by Robert Y. Al-Jaar in the *Performance Evaluation Review* of the ACM SIGMETRICS.

The author achieves the major objectives presented in his preface. Raj Jain provides computer professionals simple and straightforward performance analysis techniques in a comprehensive textbook. He gives basic modeling, simulation, measurement, experimental design, and statistical analysis background, and emphasizes and integrates the modeling and measurement aspects. The author discusses common mistakes and games in performance analysis studies, and illustrates the presented techniques using examples and case studies from the field of computer systems.

The book consists of 36 chapters organized in the following six parts: "An Overview of Performance Evaluation," "Measurement Techniques and Tools," "Probability Theory and Statistics," "Experimental Design and Analysis," "Simulation," and "Queueing Models"; nearly the same level of attention is given to each part. Each chapter has a set of carefully designed exercises; solutions to selected exercises are presented at the end of the book. Each part concludes with a comprehensive list of references, appropriately selected from the extensive list that follows the exercise solutions. The book also includes an appendix that contains statistical tables and formulas.

Part I emphasizes the importance of performance analysis for designers, administrators, and users of computer systems. The author introduces the field of computer systems performance analysis and presents examples of problems that one should be able to solve after reading the book. He discusses 22 common mistakes that occur in performance evaluation studies and presents in a "box" format a summary checklist to help avoid these mistakes. This format is an effective presentation technique used judiciously throughout the book to highlight important techniques and summarize major results. The author advocates a 10-step approach to performance analysis and discusses the selection of performance evaluation techniques and metrics.

I enjoyed reading this coverage of issues critical to the success of any performance engineering project but often ignored. The discussions remind experts of the importance of these matters and encourage newcomers to develop the correct attitude toward performance.

Part II begins with explanations of workload types. The author emphasizes several major considerations for workload selection. He then discusses monitors, including program execution monitors and accounting logs. Of particular interest is the discussion of the design of software monitors.

Capacity planning and benchmarking sections include enlightening material on common mistakes of inexperienced analysts and the games and tricks played by experienced analysts. By discussing such practical topics as load drivers and remote-terminal emulators (RTEs), the book provides comprehensive information on performance analysis, a welcome departure from the format of many other books which consider such a discussion "unintellectual." The art of data presentation techniques follows. The quality and format of the presentations in the book clearly indicate that the author does practice what he preaches.

Part II concludes with a discussion of ratio games. The author uses case studies and examples to explain how to choose an appropriate base system and ratio metric. He also outlines strategies for defending oneself from ratio games played by others.

Part III introduces the basic concepts of probability and statistics, using examples and case studies from the computer field to convince the reader that these concepts have practical importance. The author explains how to summarize measured data and use sample data to compare systems; provides an easy-to-read introduction to simple linear regression models; and discusses other regression models.

The overall treatment of experimental design and analysis is so comprehensive and thorough that Part IV is practically a short book on experimental design techniques. The author explains the basic concepts, terminology, and design techniques, and discusses in detail a variety of experimental designs.

Part V contains a good introduction to simulation as a tool for computer performance analysis. The author provides a checklist of common simulation mistakes and describes the Monte Carlo, trace-driven, and discrete-event simulation methods. Adding a discussion of process-oriented, as opposed to event-oriented, simulation methods would provide the reader with a more complete perspective of current simulation methods.

This part next describes the analysis of simulation results. Included are model verification and validation techniques, accompanied by algorithms to aid the reader in the implementation. The book also contains in-depth coverage of random number generators. Part V concludes with a brief discussion of current areas of research in simulation. Pointers to references for process- and object-oriented simulation methods would be a welcomed addition.

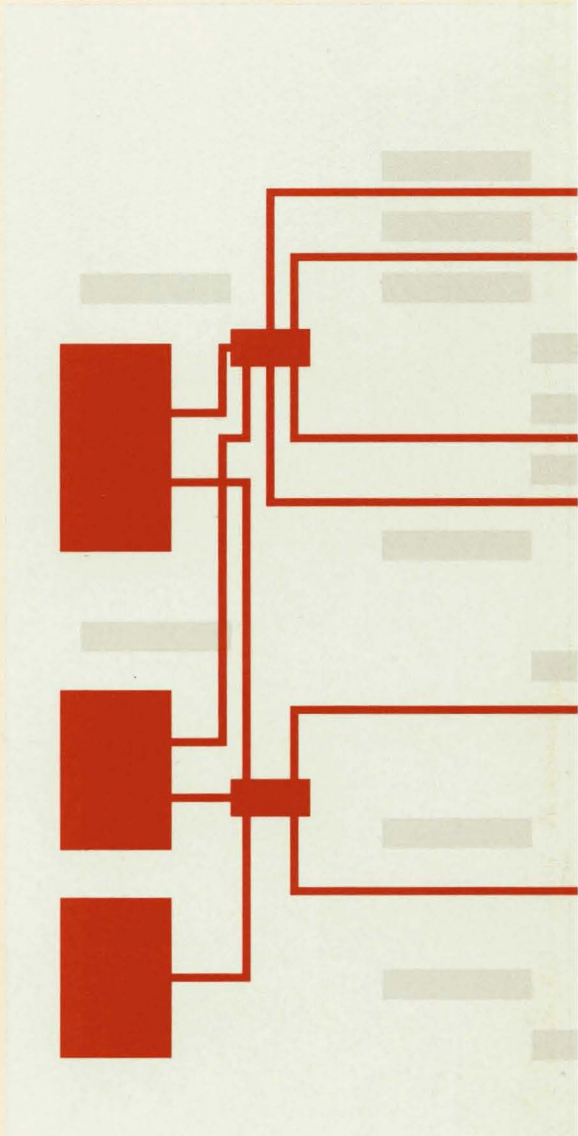
Part VI introduces the basic concepts and notation of queueing models, key tools for evaluating the performance of computer systems. Included is a clear, step-by-step analysis of single queues; a discussion of stochastic processes; an explanation of queueing networks and related operational analysis techniques; and a demonstration of the convolution algorithm. The author also introduces the reader to the practical technique of hierarchical decomposition of large queueing networks. Part VI concludes with a discussion on the limitations of queueing theory. To choose the appropriate modeling approach, analysts must be aware of these limitations.

This is a truly landmark book which achieves the author's stated objectives. A strong point of the book is its equal treatment of modeling, simulation, measurement, and experimental design in the context of computer systems. I believe that most of the chapters can be used as 45-minute lectures, as the author claims. Senior students in engineering and computer science will generally have the mathematical sophistication required to understand the material covered in this book. *The Art of Computer Systems Performance Analysis* is indeed an encyclopedia on the performance analysis of computer systems, and should be on the bookshelf of every computer professional.

Robert Y. Al-Jaar, Ph.D., Principal Systems Engineer
Porting and Performance Engineering Group
Digital Equipment Corporation
Marlborough, Massachusetts 01752-9122
July 24, 1991

Note: The book reviewed was written by an author who contributed a paper to this issue of the *Journal*. The editor included this review as one that might be of interest to our readers. The review expresses the opinions of the reviewer.

digital™



ISSN 0898-901X

Printed in U.S.A. EY-H890E-DP/91 09 02 20.0 DBP/NRO